

DOCTORAL THESIS



Engineering of IoT Automation Systems

Oscar Carlsson

Industrial Electronics

Engineering of IoT Automation Systems

Oscar Carlsson

EISLAB
Luleå University of Technology
Luleå, Sweden

Supervisors:

Jerker Delsing and Jens Eliasson

Printed by Luleå University of Technology, Graphic Production 2017

ISSN 1402-1544

ISBN 978-91-7583-791-8 (print)

ISBN 978-91-7583-792-5 (pdf)

Luleå 2017

www.ltu.se

To my family, who have always supported and inspired me...

ABSTRACT

Major societal challenges such as environmental sustainability, availability of energy and raw materials, and globalisation are creating new requirements for many actors in society. These new requirements relate to efficiency, flexibility, sustainability, and competitiveness. While these aspects have all been around for some time, and many systems have been locally optimised with regards to one or a few of these requirements, recent advances in communication and processing capabilities promise increased possibilities for connecting different parts of society, allowing optimal operation on a much larger scale.

While industrial production systems have been controlled electronically for decades, the digitisation of market channels and consumer systems, together with the possibility to interconnect different production facilities, now allow for automated interaction along the whole supply chain from raw materials to end users. Simultaneously, increased demand for efficiency forces increased specialisation among actors, which with increased possibilities of interconnectivity, creates large enterprises of cooperating, specialised stakeholders.

One of the major remaining obstacles for a widespread adaptation of more intelligent, more connected systems, able to deliver these envisioned results, is a coherent approach to the engineering and management of Systems-of-Systems involving very large numbers of devices and operating across several automation domains. For traditional automation systems there are established engineering procedures and numerous standards for engineering data, although most are focused on the static processes that have traditionally been the norm. For full integration with a digital society many of the existing automation systems will need significant modifications and as many automation systems are ageing and in need of replacement, a suitable solution to this may be a large scale migration to new automation solutions.

The work presented in this thesis includes some new approaches and methodologies to utilise the existing engineering procedures and standards, while introducing some of the flexibility proposed by the emerging technologies. The major technical solutions presented consist of a structure way to organise connected systems and how they are related, regardless of engineering standards used to design their interactions, and an approach to allow configuration of heterogeneous systems through service interactions. Further contributions include an approach for migrating certain categories of existing industrial control systems to a service oriented architecture, as a basic outline for adaption of the next generation of automation systems in industry. Certain remaining challenges have been identified, which have to be addressed for a successful launch of widespread interconnected automation systems based on Internet of Things and associated technologies.

CONTENTS

Part I	1
CHAPTER 1 – INTRODUCTION	3
1.1 Methodology	5
1.2 Thesis outline	5
CHAPTER 2 – DIGITISATION OF SOCIETY AND AUTOMATION OPPORTUNITIES	7
2.1 Systems-of-Systems	7
2.2 Automation and clouds	8
2.3 Internet of Things	10
2.4 The Arrowhead project	11
CHAPTER 3 – INDUSTRIAL AUTOMATION SYSTEMS	15
3.1 Automation systems by industrial sector	15
3.2 Industrial automation engineering	18
3.3 Standards for engineering data	21
CHAPTER 4 – ENGINEERING OF IoT SYSTEMS-OF-SYSTEMS	23
4.1 Engineering in the Arrowhead framework	24
4.2 Automation system engineering work-flow	26
CHAPTER 5 – MIGRATION OF INDUSTRIAL AUTOMATION SYSTEMS	29
5.1 Migration of industrial automation systems	29
5.2 Key differences between SCADA and DCS	32
5.3 Migrating SCADA to SOA	33
5.4 Migrating DCS to SOA	33
CHAPTER 6 – CONTRIBUTIONS TO APPENDED PAPERS	35
6.1 Additional publications	39
CHAPTER 7 – CONCLUSIONS AND FUTURE WORK	41
7.1 Conclusions	41
7.2 Future work	43
REFERENCES	45

Part II	51
PAPER A	53
1 Introduction	55
2 Challenges in migrating industrial process control systems	57
3 Migration procedure	60
4 Conclusion	65
PAPER B	69
PAPER C	77
1 Introduction	79
2 Migration	81
3 Functional aspects	85
4 Migration procedure	88
5 Migration of functionality	95
6 A first step - Migration of a lubrication system	106
7 Conclusion	118
PAPER D	125
1 Introduction	127
2 Existing standards and related work	129
3 Engineering tool interoperability	131
4 Plant description use cases	134
5 Conclusion	137
PAPER E	141
1 Introduction	143
2 Related work	145
3 Approach	147
4 Use case: Orchestration from Process & Instrumentation Diagram	151
5 Conclusions and Future work	153
PAPER F	157
1 Introduction	159
2 Related work	162
3 Proposed Approach	163
4 Use cases and Evaluation	168
5 Discussion	172
6 Conclusion	173
7 Future work	173
PAPER G	177
1 Introduction	179
2 Related work	181
3 IoT automation architecture	182

4	IoT automation engineering	184
5	Automation engineering: A process automation scenario	188
6	Application of IoT Automation Engineering	191
7	Conclusion	197
8	Future work	197
PAPER H		201
1	Architecture fundamentals	203
2	Important definitions	204
3	Documentation structure	211
4	Arrowhead Framework architecture	213
PAPER I		245
1	Introduction	247
2	Mandatory core systems and services	247
3	Automation support core systems	266
PAPER J		295
1	Introduction	297
2	Engineering of an Arrowhead compatible multi-domain facility	297
3	Component-based engineering methodology	299
4	Safety and security engineering of IoT automation systems	304
5	Engineering scenarios	321

ACKNOWLEDGMENTS

First and foremost I would like to thank my supervisors Professor Jerker Delsing and Associate Professor Jens Eliasson for the support, good advice and the unwavering confidence they have shown me, even though our offices are separated by more than seven hundred kilometers.

My colleagues at the Stockholm office of Midroc Automation, both past and present, deserve my most heartfelt appreciation for their support, their practical and technical insights and, not least, keeping it fun to go to the office.

To all of Midroc, at all locations and working positions, I would like to extend my gratitude for all the inspiration, discussions and support I have received, even though I may not always have been able to explain what I am trying to achieve with this work.

Also, a special mention must be made of my mentor, Doctor Fredrik Arrigucci, who is always so full of new ideas and was the one to give me the opportunity to pursue a PhD while still employed at Midroc Automation.

I would also like to extend my gratitude to my other colleagues, working at companies, research institutes and universities all around Europe, that I have had the good fortune to work with through the research projects IMC-AESOP and Arrowhead. In addition I would like to thank the European Commission, Artemis-JU, ECSEL-JU and Vinnova who have funded much of my work through the Arrowhead and IMC-AESOP projects, thereby making my PhD studies possible.

Last but not least I would like to thank my parents, my sister, and the rest of my family and friends for support, relief and the help to occasionally take my mind off my research.

Stockholm, January 2017
Oscar Carlsson

Part I

CHAPTER 1

Introduction

“Why bother with a cunning plan when a simple one will do?”

Terry Pratchett, Thud!, 2005

The whole world is developing ever more quickly and societies all around the globe are facing challenges of sustainability, efficiency, competitiveness, and flexibility. At a more concrete level, this takes the form of a need to develop environmentally, economically and socially sustainable societies and businesses. Competitiveness requires all producers to ensure the availability of energy and raw materials, and to use them efficiently. Additionally, global competition and communication is driving rapidly changing market trends, putting pressure on producers to adapt and giving benefits to consumers that are quick to seize new opportunities.

A most promising component in adapting to these changes is the digitisation of production, products and society as a whole. A successful large scale digitisation can enable dynamic information exchange between all actors involved. Thus allowing planning and forecasting, optimisation and adaptation on a larger scale, applicable to all parts of society, thereby building a better world for all. One branch of utilising the benefits of digitisation is the automation of new processes and improvements to already automated operations.

For Europe and more specifically for the European productive industries, further detailing of these challenges have been performed in different road maps and initiatives, such as Industry 4.0 [1], the Factory of the Future road map [2] and the ProcessIT.EU road map on process automation [3].

Three emerging trends in production industry that address these topics are:

- 1) Multi-stakeholder corporations, replacing large monolithic organisations
- 2) Learning from previous products and other parts of the value chains
- 3) Integrating information from the whole product and production process life-cycles

In particular, the Factory of the Future road map [2] and the ProcessIT.EU road map on process automation [3] both identify the need for a more integrated product and

process engineering, in addition to the already mentioned integration of the products and processes themselves.

The challenge of the automation engineering process and its associated tools can be seen as to capture a desired function, and turn it into a physical system that can be built to fulfill the intention of the designer. This is a complex task with many aspects and the most suitable engineering tools and methodology depend a lot on which part of task is perceived as the most challenging.

Most engineering disciplines and most application domains have established structured and standardised engineering tools and methods. However, because the interaction between different disciplines and domains have historically been limited and not a critical factor, the standards vary greatly and only more recently have there been major incentives in harmonising the different standards.

With the proposal of various approaches to promote integrability between automation systems from different application domains, as well as the incentives to maintain engineering data and system information along the whole life-cycle of a production process, this situation gave rise to the first two research questions.

Q1 How can run-time and design-time knowledge efficiently be integrated to manage an automation system of systems?

Q2 How can standardised engineering methodologies for capturing process and technical knowledge be utilised to manage a cross-domain IoT automation system of systems?

Furthermore, for the establishment of new technology in the European production industry, Pereira et al. highlight in their study of the European Monitoring and Control market from 2007 [4], that the three main market barriers for Monitoring and Control in the process industry are:

- Complexity of systems
- High installation costs
- Many old installations with legacy equipment

The two first observations are likely to become even more valid when the emerging trends in production and the intention to make systems from different application domains interoperable, are considered. Thus, production automation engineering and installation simplicity and efficiency becomes very important.

The last market barrier listed hints at a suitable entry point to overcome these challenges. A strong migration strategy that targets old installations would appear to have a credible chance for market success. A higher degree of digitisation should provide the possibility for increased flexibility and visibility for activities such as maintenance. At the same time, a migration strategy must look at legacy systems to see how those systems solve the challenges of complex systems with high availability and reliability. If performed well, a migration may also reduce both the complexity of the system and the

installation costs by utilizing existing structures and hardware where appropriate, thus addressing all of the three main market barriers.

These trends and observations initiated a third research question to be studied:

- Q3** What are the functional, performance and operational critical aspects of a successful migration of an industrial automation system to an IoT-based cloud solution?

1.1 Methodology

The research work presented in this thesis has been conducted in joint research projects including universities, research institutes, and industrial partners. This has brought the possibility of bringing together experience from the methods used in different parts of society.

The research methodology used has its roots in experimental computer science and engineering research, defined as “the building of, or the experimentation with or on, nontrivial hardware or software systems” [5].

For the proposal of new methods, current state-of-the-art approaches have been studied through discussions with experience engineers and researchers, through reading papers in the area, and by taking relevant courses. This has identified situations where new technology and societal developments have opened opportunities for improvement over the current state-of-the-art.

An iterative process has been used throughout this work:

1. The problem stated is analysed theoretically and a number of performance indicators for a solution are stated
2. A hypothetical solution is designed
3. The proposed design is implemented
4. The implementation is tested for the stated performance indicators
5. Return to 1 for a refined analysis

1.2 Thesis outline

This is a compilation thesis consisting of two parts. The first part describes the wider societal background, the technical context and the challenges that formed the basis for the research questions

Part II consists of five peer-reviewed papers that have been published in the proceedings of various conferences, two submitted journal papers and three book chapters. All are part of the research performed as part of this thesis work, they have been edited to follow the format of the thesis layout, but without any modifications to their contents.

The remainder of Part I is structured as follows. Chapter 2 describes digitisation, the technical concepts of Systems-of-Systems and Internet of Things (IoT), and the Arrowhead project, presenting the justification leading to the research presented in the thesis. Chapters 3 and 4 presents the specifics of engineering in the domain of industrial automation and for IoT Systems-of-Systems. Chapter 5 summarises the challenges of migration of industrial automation systems, and a proposed approach. Chapter 6 describes the research contributions of this thesis. Chapter 7 offers some conclusions drawn from the results, presents answers to the research questions, and describe directions for future studies.

CHAPTER 2

Digitisation of Society and Automation Opportunities

Large scale digitisation of society holds vast potential for improvements in many areas, including environmental, economical, and social sustainability. Some areas of society have already become increasingly digital and some of the early benefits have been reaped through automation of various functions in society.

The increasing spread of digitisation promises even greater benefits through interconnection and collaboration between ever growing, ever more numerous automated systems. One popular term for these interconnected collaborating systems is that they together constitute a **System-of-Systems**.

2.1 Systems-of-Systems

Systems-of-Systems is one of the design concepts suggested for integration of the very large automation systems proposed in road maps and initiatives such as Industry 4.0 [1], the Factory of the Future road map [2] and the ProcessIT.EU road map on process automation [3].

In particular the envisioned trend of multi-stakeholder corporations match the two defining characteristics of Systems-of-Systems, proposed by Maier [6]:

- **Operational Independence of the Elements:** If the system-of-systems is disassembled into its component systems the component systems must be able to usefully operate independently. That is, the components fulfill customer-operator purposes on their own.
- **Managerial Independence of the Elements:** The component systems not only *can* operate independently, they *do* operate independently. The component systems are separately acquired and integrated but maintain a continuing operational existence independent of the system-of-systems.

In a multi-stakeholder corporation, large monolithic organisations are becoming more modular, with independent entities where responsibilities and decision making can be distributed. As such, multi-stakeholder corporations does not necessarily have to refer to cooperating legal entities but that responsibility and decision making is distributed enough that it *could* as well be different legal entities, i.e. they are distinctly different stakeholders. Through efficient integration of the modular stakeholders, each stakeholder is enticed to become more specialised and more competitive, as the modular structure with dynamic integration allows replacement of single stakeholders to bring about a more competitive corporation.

In order to enable Systems-of-Systems that fully implement these characteristics, Maier [6] also describe four architectural design principles:

- **Stable Intermediate Forms:** A System-of-Systems designer must pay closer attention to the intermediate steps in a planned evolution. The collaborative system will take on intermediate forms *dynamically* and *without direction*, as part of its nature. Thus, careful attention must be paid to the existence and stability (in all suitable dimensions) of partial assemblages of components.
- **Policy Triage:** The System-of-Systems designer will not have coercive control over the systems configuration and evolution. This makes choosing the points at which to influence the design more important. In communication-centric systems, this means that design leverage will frequently be found in relatively abstract components (like data standards and network protocols).
- **Leverage at the Interfaces:** A System-of-Systems is defined by its interfaces. The interfaces, whether thought of as the actual physical interconnections or as higher level service abstractions, are the primary points at which the designer can exert control.
- **Ensuring Cooperation:** A System-of-Systems exists because the partially independent elements decide to collaborate. The designer must consider why they will choose to collaborate and foster those reasons in the design. This is not a consideration in the design of monolithic systems where the components can operate only as part of the whole.

Given the challenges and opportunities presented by the digitisation of society, as illustrated in the mentioned road maps and initiatives, the System-of-Systems characteristics given by Maier appears to identify and categorise some important aspects, in a generalised way.

2.2 Automation and clouds

The implementation of digital Systems-of-Systems is not generally obvious, and given the traditional methods for digital systems, the sheer scale of the Systems-of-Systems envisioned may present a significant challenge.

As a method of managing large scale digital systems, Cloud Computing has become a popular term in Information Technology (IT) for the last decade, based in a much older idea of sharing computation resources dating back to 1957 [7, 8]. The phrase Cloud Computing gained popularity in 2007 after announcements from Google [9] and Amazon [10] in the previous autumn.

As do many buzzwords, Cloud Computing has a broad meaning. In a clarifying effort the National Institute of Standards and Technology of the U.S. Department of Commerce (NIST) [11] has defined five Essential Characteristics of Cloud Computing:

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

An example of how cloud computing might appear in the field of automation systems can be taken from an Agent-based manufacturing execution system [12].

Agent-based manufacturing execution systems can utilise flexibility in a production line to allow for efficient dynamic scheduling of short production series. This is achieved through identification of units at an early stage, e.g. through an electronic tag or a small system with communication capabilities on each unit. Each identified unit can then be assigned a specialised production sequence with individual modifications. The sequence of operations and production flow is then ensured by software agents representing the units

From a functional perspective, the location of the production recipe (i.e. the data) and the execution of algorithms for production scheduling (i.e. the computation) is irrelevant. The production flow will be the same regardless of if the data and computation is centralised, with a passive tag on each unit and each operation performed on request from the centralised system, or if the units are equipped with the capabilities to communicate with each other and individually request operations at each production step through direct communication to the production systems, where the organisation is handled through a distributed algorithm. In such a scenario the location of the data and computation can be seen as obscured, as if known to exist but hidden within a cloud.

Obscurity is a question of perspective though, and only one aspect of Cloud Computing. However, as long as each unit is represented as a software agent on the networked systems, and the execution location can be changed, allowing for the essential elasticity, and it is operated through a networked, measured service-oriented approach, either implementation could be considered an implementation of Cloud Computing. Among the Essential characteristics proposed by NIST, this aspect is represented by *Resource pooling*, in that the resources are not assigned specifically by design but rather available in a pool.

Even though the general definitions of cloud computing, like the one proposed by NIST, allow for a rather wide set of implementations, the general perception of a *Cloud* has become more associated with globally accessible platforms for software execution and data storage. This is likely stemming from the implementations presented by e.g. Amazon and Google, as these coincided with the popularisation of the term.

For applications that are inherently local, in that they operate to a large degree with physical objects and measurements as points of input and output, the benefits of global accessibility are not as apparent as with more traditional IT systems. This, combined with the narrowed perception of the concept, appears to be one of the reasons for a couple of other concepts, such as *Edge* [13, 14] or *Fog* [15] computing.

For automation networks there are also several reasons to restrict the span of the resource pooling and the elasticity, e.g. to be able to predict or guarantee Quality of Service (QoS) [16], and to restrict the broad, on-demand access, e.g. to meet a certain level of safety and security according to required safety and security assessments, as discussed in the appended Paper J. These reasons have motivated the more specific concept of a *Local Automation Cloud*, as described by Delsing et al. [17].

A Local Automation Cloud, as implemented on a series of local devices running software in a distributed manner and interacting with the physical world through a number of *things* can be seen as an implementation of another popular concept, envisioned to provide countless opportunities for the improvement of efficiency, control, and comfort in society, the **Internet of Things**.

2.3 Internet of Things

The Internet of Things is envisioned to be a cornerstone in the connected, comfortable, efficient, and productive society of tomorrow. The idea behind the Internet of Things is to allow things to communicate directly with each other, e.g., to share related information from different systems and to present the information to users in a more useful manner, allowing humans to focus on decisions and actions rather than filtering and combining information from different sources.

Cisco presents a good example in an infographic [18], in which information from traffic systems, train systems and meeting schedules is aggregated and communicated to the alarm clock and car systems of an end user. These interactions exemplify the characteristics of a System-of Systems very clearly, in that all of these *things* are already independent, functioning systems, before they receive any connectivity. Through an example, such as the one by Cisco, it is possible to visualise how an SoS is formed from the independent systems, using the four architectural principles.

In the Cisco example, the alarm clock has its alarm time modified by information from a meeting schedule, a car, a traffic information system, and a train information system, resulting in the alarm clock additionally signaling the car and a coffee maker in turn. All of these systems are independent from both an operational and managerial point of view, satisfying the two defining characteristics proposed by Maier.

Furthermore, although the example is presented by Cisco as complete, it is possible to

see that any one of the connections would be a useful integration on its own, and any set of the connected systems will be a stable intermediate form. Similarly, it is obvious that the System-of-Systems designer is not likely have coercive control of such diverse systems a train information systems, a car and a coffee maker, thus satisfying the policy triage. The same applies to the characteristic referred to as “Leverage at the Interfaces”, as the capabilities in this example are fundamentally limited by the interaction capabilities of the alarm clock and of the other systems. The characteristic of “Ensuring Cooperation” does not appear clearly in the example, but it is reasonable that the described System-of-Systems exists primarily for the benefit of the user of the alarm clock, and the connections to all other systems are likely driven by commercial interest from the system providers, seeing that an owner of a connected alarm clock might be willing to pay a small premium for a connected coffee maker, and so on.

This example illustrates the benefit of interoperability between rather diverse systems, and that it is something of a requirement for widespread establishment of the IoT vision. A survey of commercial IoT frameworks and platforms, with various solutions to interoperability, was performed by Derhamy et al. [19].

Part of this vision is the Industrial Internet of Things (IIoT) whereby connected devices in industrial installations, in addition to performing the duties of the automation systems of today, provide countless opportunities for improvement thanks to dynamic access, for human and electronic decision makers alike, to systems and information that were previously obscured [20].

Cisco predicts 3.4 IP-connected devices per person on earth by 2020 [21] and that M2M connections will grow from 4.9 billion in 2015 to 12.2 billion by 2020 [22]. Only a minority of these are industry related but the Compound Annual Growth Rate (CAGR) is predicted to be over 10% for Manufacturing and Supply Chain and nearly 30% for the Energy sector.

The technology behind the Internet of Things is based on Internet Protocols (IP) and Service-Oriented Architectures (SOA) as an open platform with tested technology. Pereira et al. highlight in their study of the European Monitoring and Control market from 2007 [4], that one of the main challenges of Monitoring and Control in the fields of both manufacturing and process industries for the European market appears to be related to the establishment of IP and service-oriented architectures.

2.4 The Arrowhead project

Based on the high-level visions and road maps for societal improvements, the Arrowhead project was formed around the grand vision to “Enable collaborative automation by networked embedded devices”. Two grand challenges were identified in the proposal of the project, in the spirit of the System-of-Systems approach, focusing on the communication and interface aspects especially. The two Arrowhead Grand Challenges are:

- Enabling the interoperability of services provided by almost any device
- Enabling the integrability of services provided by almost any device

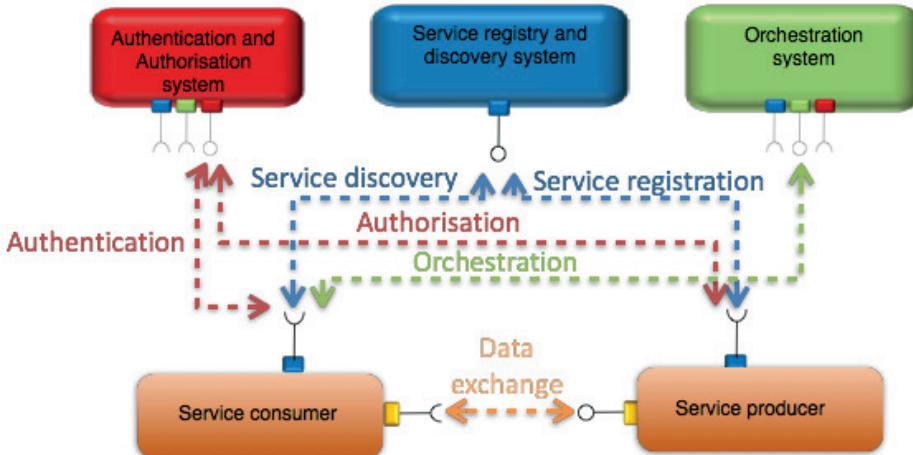


Figure 2.1: Key concepts of the Arrowhead Framework.

As previously illustrated, and further highlighted in the first of these two Grand Challenges, interoperability is of vital importance to bring the societal benefits that are envisioned for digitisation. However, interoperability is a wide concept and can come in different varieties, both in terms of the technical level that can be agreed upon but also what span of functionality that is included. E.g., as Derhamy et al. [23] illustrate in their proposal for handling error-messages during translation between communication protocols, a more complete approach to interoperability must handle much more than simply handing over a data package between two devices.

Ultimately, interoperability will have to depend on the application and the situational requirements, but it is likely that for the scope targeted by Arrowhead, the challenge will include the following interoperability aspects:

- Protocol interoperability
- Security interoperability
- Semantics interoperability

All of these aspects have been investigated during the project, with the first one showing the most promising results, summarised by Derhamy [23, 24, 25]. Security interoperability has been addressed within the project, but significant challenges remain [25, 26]. Simultaneously, significant progress has been made by other research groups on the challenge of Semantic interoperability, although much work remains to be done [27, 28, 29].

The second Grand Challenge of integrability of services illustrate that through integration by service interactions, a greater function can be fulfilled, much in the same sense that Systems-of-Systems may be able to address the greater societal challenges in a way that the independent systems have yet been unable to.

The cornerstones for enabling integration of systems will be to understand 1) what things there are that could possibly be integrated, and 2) how these things should be connected so that the integration leads to added functionality or value.

From the Arrowhead project has emerged the Arrowhead Framework for integration of interoperable systems, through the use of service interactions [26, 30]. Within this framework, the method for finding other systems is through a Service registry and discovery system, and the method for finding how systems should be connected for greater benefit is through an Orchestration system.

Given the openness of a framework based on the principles of a service-oriented architecture, it is additionally deemed necessary to have a trusted actor to certify the authenticity of connected systems and authorise service interactions in the System-of-Systems, a function that in the Arrowhead framework is provided by a Authentication and Authorisation system.

These three core systems for the basis of the Arrowhead framework, by enabling the interoperability and integrability between compatible connected systems, as illustrated in Figure 2.1.

Some of the key scientific contributions presented in this thesis represents the approach by which a System-of-Systems based on the Arrowhead framework can be designed to fulfill a desired function, through populating the Orchestration system.

CHAPTER 3

Industrial Automation Systems

The previous chapters discussed the most modern technologies that society is currently discussing and developing. To better understand the environments such systems have to operate in, it can be helpful to first study the history of what we today consider legacy automation systems.

Industrial control systems can be traced back to the very first industrial machines in the 18th century, through famous scientists and inventors such as James Watt, William Siemens and James Clerk Maxwell, from purely mechanical systems to electrical relay systems and further on to microprocessor based systems in the early 1970's [31].

Today virtually all industrial control systems are based on microprocessor systems, small electronic devices distributed across all areas of industrial systems. Although the electronic hardware today is not so different in the different areas of industrial automation, there are significant differences in the structure and their organisation. Through these technical differences and through some historical differences between industrial sectors, many different approaches have been established for engineering of the different industrial automation systems.

3.1 Automation systems by industrial sector

Throughout the evolution of industries and industrial control systems, the focus and challenges have differed between industrial sectors. Generally speaking, there can be said to be three main categories of industries affecting the automation systems that have evolved in these sectors, these are:

1. **Manufacturing industries** where products are made piecewise, often through the assembly of components into a complex product ready for delivery to a consumer market. The production flow is characterised by a series of machines along a transport system, where products arrive one at a time to each machine which performs a series of operations on each product before the product is transported to the next machine.

2. **Process industries** in which a raw material is processed into a more refined material, often for delivery to manufacturing industries. The production flow is usually continuous, going through highly specialised machines that process the material as it flows through or captures in large vats for processing before it is released in a continuous flow to the next machine. For automation purposes, large energy production facilities such as power stations are often grouped with process industries, as a facility where a raw material such as coal, oil or waste material is processed into heat and electric energy.
3. **Utilities and infrastructure:** Although technically not production facilities, utility systems such as water and sewage networks, power grid systems, rail networks and tunnel systems are often considered industrial systems. These networks are all geographically distributed, with some powered systems spread out over great distances, with some need to monitor the systems and with a need to perform some local control actions at the widespread locations.

Through the practical differences in operation, different markets and different technical evolutions, these three industrial sectors have developed different automation traditions and approaches to the engineering of these systems.

3.1.1 Manufacturing industry automation systems

Mehrabi et al. [32] identify four historical manufacturing paradigms (Mass production, Lean manufacturing, Flexible manufacturing systems, and Reconfigurable manufacturing systems), up until the year 2000. Each of these paradigms being focused on addressing a very specific competitive advantage, and each new paradigm depending on or including the previous ones. In a similar manner Jovane et al. [33] identify Craft Production starting in the middle of the 19th century, Mass Production starting with the invention of the assembly line in 1913, Flexible Production introduced in the 1970's, and Mass Customisation and Personalisation starting around the year 2000, and predict a fifth paradigm in Sustainable Production for the 2020's. Figure 3.1 illustrate the historic progression in terms of variety and volume.

The manufacturing industry has traditionally used a lot of manual labour, since the start of the era of mass production, organised at stations along the assembly line, and the automation of manufacturing has until all but recently been intended to improve the efficiency and quality of the traditional assembly line. As such, the general approach has been to automate the operation at each station individually, resulting in relatively independent automation at each station, with only limited monitoring and coordination between stations through a supervisory system. This set-up is called a PLC-SCADA approach, using a Programmable Logic Controller (PLC) at each station and a Supervisory Control And Data Acquisition (SCADA) system for supervision and coordination.

This separation between local and distributed control means that each station or production cell can be engineered more independently, allowing suppliers of manufacturing equipment to deliver fully automated production cells, only providing a limited communication interface to the SCADA. This lowers the cost of acquiring each production

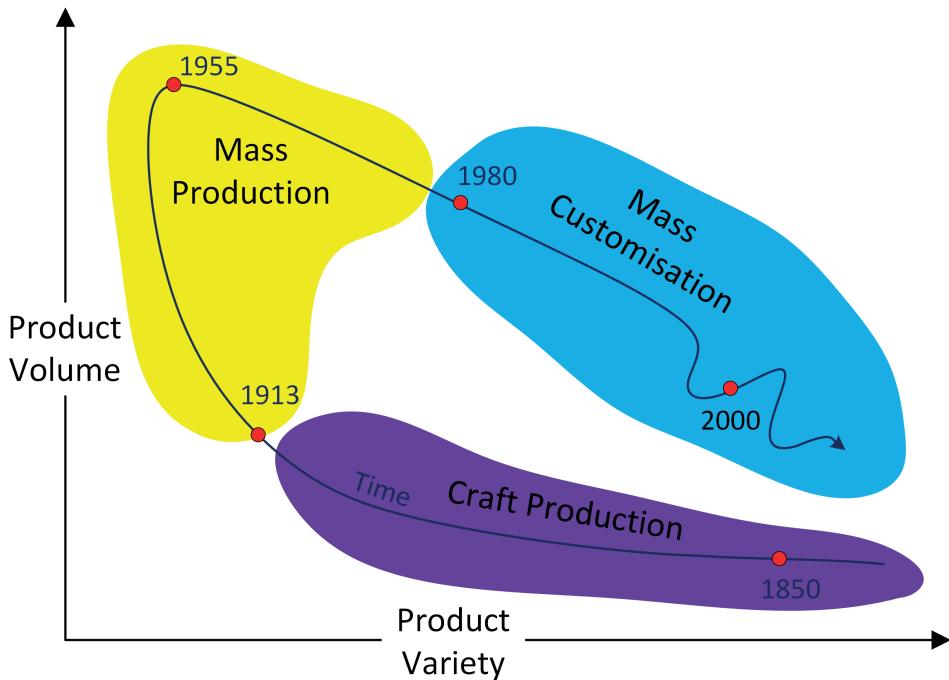


Figure 3.1: Illustration of the historic progression of manufacturing. Adapted from [33].

cell but can lead to complications during operation and maintenance of the systems, if production cells have been acquired from different suppliers not following the same engineering practices.

3.1.2 Process industry automation systems

Compared to the manufacturing paradigms, process industries were for a long time focused exclusively on product cost and product quality; the significant competitive advantages identified by Mehrabi et al. [32] as significant for Mass production and Lean production, respectively. This has most likely contributed to the affinity for highly specialised equipment with very high reliability and throughput within the sector. Only more recently has there been an interest in more customised products from the process industry, possibly as a result of globalisation where the largest and most efficient plants are able to serve the entire global market of bulk products forcing smaller plants, with higher relative operating costs, to go out of business or focus on specialised markets where the revenue per volume is higher.

Partially due to this focus on efficiency and quality in the production, and partially due to the high investments made in the process machinery, the automation in process

industries has developed in a different direction than that of the manufacturing industries. In process industries there is a strong focus on the process flow, throughout the whole production facility, and as such the automation is organised as one tightly connected system, with control execution hardware distributed throughout the facility. This is referred to as a Distributed Control System (DCS). While the electronic hardware in each of the controllers in a DCS can be very similar to a PLC, a major difference is that the engineering of the whole DCS is done centrally, ensuring that the same engineering approach is used across the whole facility but often at a higher cost as it is more difficult for suppliers to streamline their engineering procedures.

3.1.3 Utilities and infrastructure automation systems

In the field of utility services, such as water and sewage transport, and infrastructure, such as railroad networks, power grids, motor highways and tunnels, the automation challenges have long been focused on immediate local control and long-distance supervision. The commercial situation is usually very different compared to the two other categories, with utilities and infrastructure systems often serving a specific geographic area with very little competition within that area. As the services are often funded, owned or heavily regulated by governmental or municipal bodies, the focus is often a required level of service at minimal cost. Therefore, there has been an even stronger focus on cost than for process industries, and only very limited concern for quality.

In this environment the automation systems have taken on a third type of structure, called SCADA systems as is common in manufacturing industries but traditionally with Remote Terminal Units (RTU) executing the local control. An RTU is similar to a PLC but the structure is much more focused on the communication issues that may occur in such geographically distributed systems. Given that continuous, reliable operation is critical in this area, an RTU is expected to be able to continue its operation even if the connection to the supervisory system is disabled for some time, and usually configured to store all data locally until the communication can be resumed. Compared to the operation of a PLC in a manufacturing scenario, the control scenario for an RTU is usually much simpler with only one mode of operation controlling a few devices.

3.2 Industrial automation engineering

Much of the industrial automation engineering is performed as projects and can be structured using a traditional project management approach using the five phases of a project that are illustrated in Figure 3.2. In Paper [G? - fix!], a more specific work-flow for automation engineering is described, following the five steps:

1. **Conceptual application design:** This step is comparatively informal and usually not technically specific but should outline the purpose and motivation for the application. Typically it should answer the question “**Why?**” and provide basic requirements to the design, given available resources and other constraints.

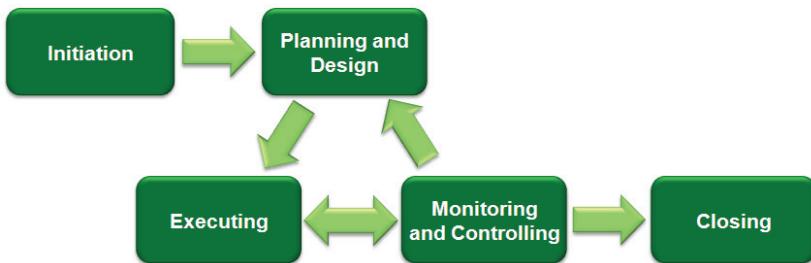


Figure 3.2: Project Management main phases. Illustration from Wikimedia commons, licenced under GNU Free Documentation License.

2. **Functional design:** This step will produce most of the overarching design documents. The results of this should be a detailed design of the mission critical functionality and functional requirements for the support systems.
3. **Procurement and Engineering:** These two activities are often performed in parallel, and will affect each other. Certain requirements and design decisions may limit procurement to a single option, that will affect engineering. Subsequently, engineering decisions may pose additional requirements on procurement of other parts or subsystems.
4. **Deployment and Commissioning:** As all equipment is purchased, configured for the application, and delivered to the site of operation, a process of deployment and integration of all systems begin. When systems have been connected a series of integration tests and commissioning starts, after which the full, interconnected system-of-systems is ready for operation by the end user.
5. **Operation:** When the operational phase starts, the full system-of-systems *should* perform at the requested level without any need for further engineering. However, in most scenarios there are minor details that were not foreseen during the design phases and must be adjusted. Similarly, external factors, such as markets or regulation, are likely to change during the operation of a large system. All of these may require updates to design and engineering, including documentation and data.

The work-flow serves as a basis for the engineering of automation systems based on Internet of Things, but it is based on an amalgamation of the work-flow used for traditional engineering of automation systems in various domains.

3.2.1 Engineering similarities and differences

Due to the aforementioned differences, the automation engineering within these three categories has developed in slightly different directions.

In the Manufacturing industry, there is a high acceptance for specialised controllers for tasks such as robot control and Computerised Numerical Control (CNC) for, e.g.,

automated cutting, welding, and grinding. With time, some of these controllers have expanded their functionality to encompass the full control of a manufacturing cell, including presenting an interface that can be directly integrated to e.g. a SCADA system for organisation at a higher level. In some applications, there is still common practice of instead having a PLC in control of each manufacturing cell, interacting with and co-ordinating the specialised controllers within the cell, and communicating with supervisory systems. In both cases, the engineering of the control at the level of the specialised controllers is usually based on 2D or 3D geometric models or drawings that are interpreted by the specialised controllers into movement patterns, with the interpretation specific to the machine builder. The coordination between different machines is usually based on sequential coordination, for which programming in Sequential Function Charts (SFC) is well suited, SFC being one of the traditional PLC programming languages described by IEC 61131-3 [34].

Comparatively, Process industries typically prefer to have as much of the control integrated directly in the Distributed Control System (DCS). This brings with it a number of advantages, to which the owners, operators and engineers in the industry have now become accustomed. From an engineering perspective, a significant advantage is that much more of the systems are accessible from a single engineering station using a single engineering environment. On the other hand, the investment in automation is typically more costly, as it is more difficult to re-use engineering effort from other production plants as the production segments are not as clearly separated into single units as is customary with manufacturing cells.

As the process industries are typically not concerned with the geometrical properties of their products but rather have different quality aspects, the engineering has not been as focused on the usage of product models in the engineering. To the extent that models are used, they are usually more of simulations of various thermal and chemical processes, visualised in graphs or symbols. As most process operations are not shifting between different states according to a sequence, but rather operate continuously with some variance around the preferred optimum, the programming is preferably done in Continuous Function Charts (CFC), an extension of the traditional PLC programming languages described by IEC 61131-3 [35, 36].

For utility networks, the previously mentioned structure of the automation systems, the commercial focus on cost reduction and the situation where many control points along a network are practically identical, has led to an engineering strategy where each control point is engineered, programmed and built exactly identical. As this means that the design can be reused many times using the exact same combination of components, the engineering effort is spread on a very large number of systems. This subsequently means that the engineering cost per system is relatively low, compared to both manufacturing and process industries. On the other hand, the relatively simple control points and geographic distances has pushed more functionality to the SCADA level, especially in the areas of remote monitoring.

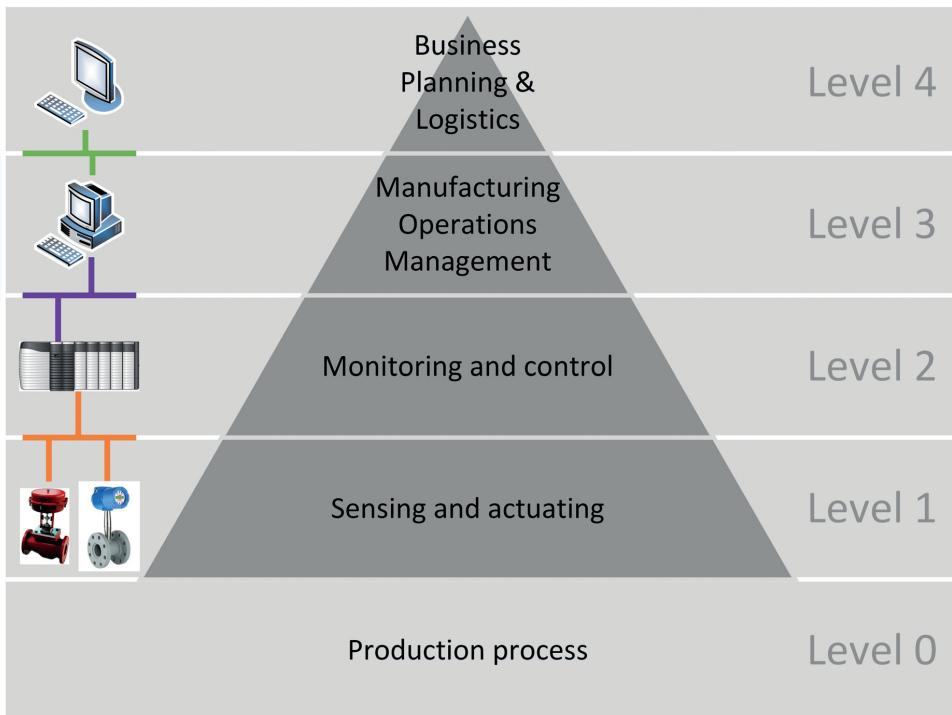


Figure 3.3: Functional hierarchy as defined according to ISA-95 / IEC 62264. [37]

3.3 Standards for engineering data

In spite of the differences, virtually all traditional automation systems within both manufacturing and process industries are organised on a higher level according to the standard IEC 62264, based on ANSI/ISA-95. This standard describes the hierarchy between systems, as illustrated in Figure 3.3, and prescribes limited interaction between systems at different levels of the hierarchy. In terms of the previously described DCS and PLC-SCADA solutions, they all reside within level 1 and 2 of this hierarchy.

Similarly, at the other end of the scale, most automation programming today is implemented in one of the five languages defined by the standard IEC 61131-3 [38], or an extension thereof, such as Continuous Function Charts.

At the same time, at various levels from the general view of the ISA-95 hierarchy to the implementation details of IEC-61131, there is a variety of competing and complementary standards for system management at various stages of the industrial automation life-cycle, including e.g. design, engineering, operation and maintenance.

Fernández and Márquez [39] illustrate some of the differences between the standards promoted by the IEEE Utility Communications Architecture (UCA®) International Users

Group, which works closely with the IEC Technical Committee (TC) 57 IEC 61850, IEC 61970 and IEC 61968, primarily associated with power systems management. And on the other hand, ISO TC 184, collaborating with “Machinery Information Management Open Systems Alliance” (MIMOSA), promotes standards primarily associated with the category above described as process industry. These standards include, e.g., OPC (“OLE for Process Control”), ISO 15926 - Industrial automation systems and integration, and ISO 18435 - Industrial automation systems and integration – Diagnostics, capability assessment and maintenance applications integration.

For the manufacturing industry, the German initiative Industrie 4.0 promotes the following standards in the Reference Architectural Model Industrie 4.0 (RAMI4.0) [40]: For life-cycle and hierarchical structure:

- IEC 62890 “Life-cycle management for systems and products used in industrial-process measurement, control and automation”
- IEC 62264 (ISA-95) / IEC 61512 (ISA-88)

For end-to-end engineering:

- AutomationML
- ProSTEP iViP
- eCl@ss

Additionally, the standard IEC 81346 “Industrial systems structuring principles” [41] is commonly used in both manufacturing and process industries.

There are several standards used in more than one industrial domain, and there are already some initiatives on synchronization between pairs of complementing standards such as collaboration between AutomationML (IEC 62714) and OPC-UA (IEC 62541) [42] as one example and collaboration between ISO 15926 and Mimosa [43] as another. Also, Göring and Fay [44] illustrate how IEC 81346 can complement AutomationML, but across the different automation domains there are still many different standards available, most with some significant advantage within their most prominent domain.

CHAPTER 4

Engineering of IoT Systems-of-Systems

To form a functioning System-of-Systems out of element systems based on Internet of Things as discussed in Chapter 2, a coherent approach, such as an IoT Framework, is required.

Within the scope of using Internet of Things (IoT) devices in a Service-Oriented Architecture (SOA) to create inter-domain Systems-of-Systems (SoS) for automation, there are some limiting factors to the methods available for constructing the full System-of-Systems. Looking at the principles for IoT, SOA, and SoS, it is clear that the SoS coordination should have as little impact as possible on the design of the individual systems. Given the architecture, it can also be assumed that the interaction should be through services.

Furthermore, relating to the envisioned scenarios with multiple cooperating stakeholders within one automation application, it is likely that some stakeholders will have the possibility to change how an individual system operates while some others will have the authority to decide how systems should interconnect to fulfill the automation application. This may be the case both during design-time, including the engineering work, and during run-time. The effects of multiple cooperating stakeholders has not yet been investigated, but remain an area of potential future work, as discussed in Section 7.2.

Given the envisioned number of units that are to be engineered in this fashion, it is also important that the engineering is very efficient, which has been an important factor in this thesis. To limit the scope and refine the solutions, the concept of *engineering* in this area can be detailed further. In essence, the engineering tasks addressed in this work can be sorted into addressing one of the two questions:

1. How should each system be set to act as the controlling stakeholder desires?
2. How should cooperating systems be made to do what a System-of-Systems application designer intends?

In general, the phrasing used here is that the first question is handled through Configuration while the second one is handled by Orchestration. These names are the ones primarily used in the Arrowhead framework [26, 30], but similar concepts should be available for other IoT frameworks and platforms as well. The rest of this chapter will present some methods and approaches to address these challenges in an efficient manner that can be used for virtually all systems, across all of the automation application domains targeted by Arrowhead and hopefully others as well.

4.1 Engineering in the Arrowhead framework

Three Arrowhead core systems were presented in Section 2.4; Authorisation and Authentication, Service registry and Orchestration systems. These three systems are considered mandatory core systems of the Arrowhead framework [26, 30]. The design also contains nine automation support core systems, as detailed in the appended Papers H and I. For the engineering methods presented in this thesis, five of the Arrowhead core systems are of particular interest:

- **The PlantDescription system** is intended to provide a basic common understanding of the layout of a System-of-Systems, and to act as an intermediate abstraction layer, based on existing standards for design and engineering data, isolating only the objects and their relations. One key usage of the Plant Description is as a source of design data for the orchestrator [45, 46].
- **The Configuration system** is designed to provide a generic approach to management of configurable devices and systems, the Configuration system provides a service acting as an organised storage for configuration files [47].
- **The Orchestration system** is responsible to assign service providers to consuming systems. Several implementations have been made, as described by Hegedűs et al. [48], but with regards to the engineering work-flow these are interchangeable.
- **The System registry** is designed to provide a local cloud storage, holding the information on which Systems are registered with a local cloud, meta data of these registered System and the services these systems are designed to consume [26].
- **The Device registry** shall provide a local cloud storage, holding the information of which Devices are registered with a local cloud [26].

Considering that the work presented in this thesis was intended to support the Arrowhead project, which spans not only the three mentioned industrial domains (manufacturing, process, and utilities) but also building automation which uses several other standards, the structuring and identification of systems was considered a significant challenge to the objective of developing a coherent engineering and operation methodology, usable by most of the targeted domains. The following list was composed as a collection of important standards, each relevant to one or more of the targeted domains:

- CAEX (IEC 62424) [49]
- AutomationML [50] (IEC 62714)
- MIMOSA [51]
- OPC UA [52] (IEC 62541)
- IEC 61499 Function blocks [53]
- IEC 81346 Industrial systems structuring principles [41]
- fastAPI Swedish building automation [54]
- IEC 61850 Power utility automation [55]
- ISO 16484 Building automation and control systems (BACS) [56]

It is well recognised that this is not a complete or exhaustive list, but it is hoped that the list is large and diverse enough to expose the most critical issues that may appear in creating a coherent approach for so many different application domains.

The integration of data from several standards is not a novel challenge, in fact there is already an available standard for integration of data from two or more standards, the standard ISO/TS 18876 “Industrial automation systems and integration – Integration of industrial data for exchange, access and sharing” [57]. This standard provides a methodology intended to capture all of the data of two or more sources that are using different standards, and create a new data-set including all of the captured data. This is a useful approach on an application basis, where each data source does most likely not use all of the possibilities of each standard, or when it is absolutely necessary to merge two sets of data. However, as a general approach for engineering and management of multi-domain automation systems, it is likely to become cumbersome, as each added standard is almost certain to grow the data-set.

In an attempt to enable cooperation in engineering between users of these different standards, the Plant Description is proposed as a unifying concept. Using the idea that all domains identify objects or systems as units that are related to each other, it is possible to record the identities of them and their respective relations. The details of the concept is presented in the Papers D, E, H and I, with further examples of its usage in Papers G and J.

For the configuration of automation devices, there are not as many standards in common use, but more providers use proprietary formats. In order to present an approach to configuration that could incorporate the variety of open, standardised and proprietary solutions, the Arrowhead Configuration system was designed to keep track of and transfer configurations. The approach is presented in Papers F, H and I, with some examples of its usage in Papers G and J.

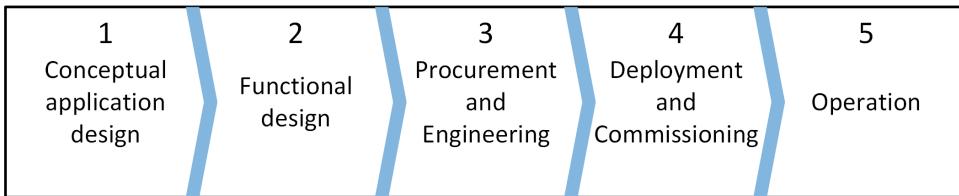


Figure 4.1: Five steps of an automation system engineering work-flow.

4.2 Automation system engineering work-flow

In Paper G, an engineering work-flow is presented, encompassing five steps from Conceptual application design to Operation, as illustrated in Figure 4.1. The work presented in this chapter is primarily focused on the third and fourth of these five steps, “Procurement and Engineering” and “Deployment and Commissioning”.

Starting from Conceptual application design, there are no proposed additions or changes for this phase. While there is great potential for new applications using the new technology, it is assumed that the design process at this level will remain essentially unchanged, as there is very little specification at technology level involved at this stage.

Regarding Functional design, step two of the work-flow, it is likely that there would be additions to the process due to new possibilities, e.g., the late binding between systems that is offered in a Service-Oriented Architecture. However, the work presented here is more focused on the engineering, as in turning a design into a working system, rather than on how to best capture the design in the first place.

For the two connected tasks of *Procurement* and *Engineering*, there are some significant differences proposed, compared to current practice. Given the increased independence of IoT devices in a system-of-systems approach, there should be better options for Procurement to focus on performance and commercial concerns, with less effort spent on assuring interoperability.

For the task of *Engineering* in this step, the work-flow proposes that an engineer can decide which types of automation devices to use based on documentation such as the System Descriptions (SysD) and Service Descriptions (SD) used in the Arrowhead framework [26, 58]. For devices that need to be configured for the intended application, configuration files can be constructed offline and organised using the Configuration system, without direct access to the device. This is further described and discussed in Paper F.

A second significant proposal for the engineering task is usage of the Plant Description concept, described briefly in the previous section. As part of the work-flow, the Plant Description is used both as an abstraction of the system-of-systems, to be used by other engineering systems and user interfaces, and as input data to create orchestrations in the system-of-systems. The concept of the Plant description and its potential usage in engineering is further described and discussed in Papers D, E and G.

The fourth step of the work-flow, Deployment and Commissioning, is intended to use

the information stored in core systems, such as the PlantDescription system, the Configuration system and the Orchestration system, to streamline primarily the deployment, as described in Papers F, G and J. During deployment, the PlantDescription system is intended to help in the identification of devices and the Configuration system reduces the number of manual operations. During commissioning, the Plant Description should prove useful as a tool for System-of-Systems overview, similarly to the usage scenario proposed for automotive industries in Paper D, where the Plant Description serves as a point of reference when different viewpoints or data sources need to be related to each other.

Operation is the final step of this work-flow, though the life-cycle of most of the systems and data continues. There are several aspects of the methods included in the work-flow that can benefit operation and later phases of the life-cycle. Some of these are described in Paper J, but they are not included in the work-flow and thus not discussed in detail in Paper G.

CHAPTER 5

Migration of Industrial Automation Systems

A very specific type of engineering projects concern the situation where an existing system is already providing much of the desired functionality, and the intention is that the new system should replace and extend the functionality provided by the existing system. This group of engineering projects is called migration projects and the key trigger for such a project is usually a desire to replace the existing technology with a whole new generation of technology, or a new paradigm.

5.1 Migration of industrial automation systems

There is a long tradition of migration of Information Systems (IS) within the field of Information Technology (IT) and computer science [59, 60]. Compared to a generic IT-system migration, the procedure for migrating an automation system does have some similarities that can be used to draw conclusions or at least form a hypothesis. One similarity between the two is that they both have a core of raw data and usually several different user interfaces, and in between these, there are one or more layers of subsystems that combine and process the raw data into more accessible information before it is presented for the user to act upon.

However, there are also significant differences between IT systems and industrial automation systems, most significant are the direct connections from an automation system to the mechanical components, sensors and actuators that are part of the physical process. This means that there is an added level of complexity in migrating the sensors and actuators, compared to the databases or similar systems of the IT world, and this complexity can carry over to the layer of systems and applications that access those components.

To facilitate a general migration strategy using migration paths as described by Delsing et al. [61] the IMC-AESOP project [62, 63, 64] provided a set of key issues that should be addressed in any migration plan for an industrial process control systems.

1. Current situation
 - (a) Analysis of the legacy systems including state of documentation, interfaces and tools.
 - (b) Identification of systems characteristics and parts that should/could be migrated. This is driven by internal strategies, available solutions and current KPIs
2. Desired situation
 - (a) Identification of business needs, requirements and goals
3. Transition/Implementation phase
 - (a) Evaluation of possible migration paths and choice of the most suitable ones
 - (b) Identification and evaluation of cross-relations between migration paths for different KPIs
 - (c) Identification of needed migration tools
 - (d) Process migration steps and check-tests
 - (e) Execution of the migration and stepwise evaluation
4. Risk analysis and Risk mitigation strategies
 - (a) Risk analysis and impact
 - (b) Risk mitigation strategies
5. Measures and trends
 - (a) Evaluation and verification of the overall result (both at the system and process level)
 - (b) Further optimization towards KPIs, e.g., reliability, performance, etc.

Compared to the five phases suggested by Bisbal et al. [59] (1. Justification, 2. Legacy System Understanding, 3. Target System Development, 4. Testing, 5. Migration), and the project management phases illustrated in Figure 3.2, there are some links to be found. The first is from the Desired situation to Justification, which is part of the Initiation and provides part of the input to Planning and Design. From the Current situation there is a link to Legacy System Understanding, which is another requirement for Planning and Design. The Transition/Implementation phase is focused on the plan for Migration, i.e., the result of Planning and Design that is used for Executing. In the end, Risk analysis and Risk mitigation strategies together with Measures and trends provide the basis for Testing or a plan for Monitoring and Controlling.

All of these strategies and approaches have a common thread; however, there are some differences, as for example, that these last key issues listed here are not strictly different phases but aspects that should be considered for the migration process.

A general migration procedure for industrial automation systems could look something like this:

1. Justification

Including the desired improvements, new and existing requirements, conditions and extent of the migration.

2. Select target solution

This process must take all technical, organizational, and business considerations into account, such as maintainability, future availability of know-how, hardware and software support, and all requirements that were identified in the justification process.

3. Find step-size for step-wise migration

This will define much of the migration strategy. As O'Brien and Woll [65] mentions there are cases where an all-at-once strategy is advisable but this is likely to be more rare the more modular both the existing and the target systems are.

4. Identify system modules for migration

This process should be based on knowledge of the existing system and may be based on hierarchy, subsystems, hardware or software objects, functionality or other concepts but it should always take the outcome of justification, target solution and migration step-size into account.

5. Organize system modules into migration steps

At this stage it is vital to verify that all required functionality is available throughout the migration process, meaning that special consideration must be taken to verify that all required interconnections between system modules are available before and after each migration step.

6. Plan, execute and validate each migration step

Each individual step will require some more detailed analysis of the legacy system, target system development, testing, commissioning and validation.

7. Monitor and validate

Before closing the migration project the situation as a whole should be documented and validated with regards to the justification and requirements that were set at the beginning.

8. Close the project

Evaluate the migration as a whole and try to find points for improvement of the process that can be used in the future and shared with others.

The above process describes the considerations made when the strategy for migration of ISA-95 structured Distributed Control Systems to a Service-Oriented Architecture presented in paper C was developed. However, as the justification was somewhat forced and the target solution was already defined in that scenario it was necessary to make

some assumptions. When the process was used in the demonstration presented in paper B it was halted in the middle of **Plan, execute and validate each migration step** with the first step executed and validated but the following migration steps only planned at a higher level.

As discussed in Chapter 3, there are a few different structures by which traditional industrial automation systems are built and organised. The one typically used in process industries, the Distributed Control System (DCS), is often built, configured, and operated as one strongly interconnected system. Therefore, this type of industrial automation systems can be seen as a very interesting example for the proposal of a migration procedure, intended to facilitate the migration from a traditional automation system to a System-of-Systems based on a Service-Oriented Architecture (SOA).

When discussing the migration of industrial process control systems structured according to the ISA-95 as in figure 3.3 it is worth noting that Supervisory Control And Data Acquisition (SCADA) systems are usually considered to represent level 2 in the ISA-95 structure with Remote Terminal Units (RTU) or Programmable Logic Controllers (PLC) acting as local controllers in level 1. Distributed Control Systems (DCS) on the other hand are usually considered to span both level 1 and 2, sometimes including functionality otherwise provided by level 3 systems. Higher level systems are usually not considered part of the control system but rather connected to and interacting with it.

Although the difference in technical capabilities between SCADA/PLC solutions and DCS have diminished over the years, there are still certain operational and architectural differences that makes the process for migrating a DCS very different from migrating SCADA/PLC-based systems.

5.2 Key differences between SCADA and DCS

Most of the differences between SCADA and DCS can be traced to the differences in design goals and for most systems today it is possible to do everything with both types of systems, but one is often simpler, cheaper or better suited for each case.

In essence, it can be said that a SCADA is data-gathering oriented, while a DCS is process oriented. The controllers or PLCs in a DCS are characterized more by distributed execution platforms rather than independent systems, and for that purpose, the communication networks within a DCS are often built redundantly so that communication is always available.

In contrast, the local controllers in a SCADA configuration are designed with possible interruptions in communication in mind, meaning that they are much more independent and usually designed to continue their operation for some time without a connection to the SCADA. In some cases, the system uses a Remote Terminal Unit (RTU) that manages the communication to the central supervisory system, while in other setups, this may be performed by components integrated in the PLC responsible for local control.

A SCADA solution is more often used in scenarios in which all information required for normal operation is available locally, such as long distance utility networks or factory automation in which each station can perform its task independently of other stations.

The DCS solution is more often used in scenarios in which communication between the local controllers is critical for operation, such as when the steps in an industrial process are more of a series of systems operating in a continuous flow or even a short halt in the process flow could cause serious accidents, meaning that the whole process has to be stopped if any one station is unavailable even for a short period.

5.3 Migrating SCADA to SOA

As a SCADA system is more focused on communication and accessing information from different systems, it is generally capable of using a number of different protocols and communication standards.

Some SCADA RTUs, e.g., the ones used by Nabil and Mohamed [66], already have web services implemented, and as the units often are truly distributed, such a system should be a relatively easy to migrate to a service-oriented architecture as the system can be more easily broken down into components that can be migrated one by one. Still, there are many similarities between SCADA and DCS that may be used to draw some conclusions from one case to another.

Gilgor and Tunc [67] mention in their paper describing a service-oriented SCADA system that migrating from an old system to their proposed system is possible using a gradual migration approach.

Migrating a SCADA system according to the strategy outlined in chapter 5.1 would in most circumstances be a straightforward gradual migration where the SCADA supervisory system is migrated in as one system module and each RTU or PLC is considered its own module. For the migration to be a smooth process it would be required that some of the migrated modules have backwards compatibility, and most likely this would be the supervisory system, or use mediators or similar technology to handle communication between the migrated and the non-migrated system modules.

If some system modules can be migrated to backwards compatible systems it should be a good choice to migrate these systems in the first migration step(s) and subsequently migrate the remaining parts as required. As most SCADA systems are quite versatile in compatibility with different protocols and technologies it should be possible to migrate the supervisory systems to backwards compatible systems first and later migrate RTUs and PLCs.

5.4 Migrating DCS to SOA

Although most commercial DCS today provide functionality that is somewhere between object oriented and service oriented, with function blocks and control modules for abstraction and reusability, they usually have more and tighter connections over proprietary links than a comparable SCADA would have.

Suppliers of Distributed Control Systems long had the commercial strategy to keep the core parts of the system proprietary and closed to competitors. This makes the

system more easy to work with as long as all components are from the same supplier, but in many cases, it has resulted in plants in which different sections of the same plant use very different systems, which causes issues both from a technical perspective but even more so from a maintenance, operation and engineering perspective, as many different qualifications are needed and resources are more difficult to share.

The migration strategy proposed in Paper A and C breaks a tightly interconnected system, which is a characteristic of Distributed Control Systems, into manageable pieces. The strategy with four distinct migration steps and mediator technology allows full operational functionality throughout the migration process. This is an example of how the strategy presented in Chapter 5.1 could be used in a less generic scenario. In this case the scenario is limited to the migration of a modern DCS to the IMC-AESOP architecture described by Karnouskos et al. [68]. In this scenario it is also assumed that the systems at level 3 and 4 of the ISA-95 pyramid are already SOA compatible and only require minor modifications, if any.

With the use case demonstration at LKAB, presented in Paper B, a first step in this strategy has been shown to be technically viable and even with prototype technology the commissioning could be performed in a timely manner during a scheduled maintenance stop at the plant, showing that a migration could potentially be performed with minimal interruption to the production process. This demonstration also followed the approach described in Chapter 5.1 but as the demonstration only covers the first migration step it could also be said to follow a traditional project management approach of Planning, Executing and Monitoring.

However, the complete migration strategy proposed by the IMC-AESOP project also requires two much more complex mediators than the one used in the demonstration for step two (Configuration) and step three (Data processing).

The second step proposes the migration of all configuration management resources, traditionally represented by the Engineering station in a DCS, whereby staff with the required authorisation are able to make significant changes to the DCS. This will require a mediator that is able to translate the functionality that is required to configure and manage the systems not yet migrated.

The third step proposes the migration of all systems that are not involved in fast control loops operating in the millisecond range. The mediator required at this point will have to manage a vast data-flow at a reasonable speed to be able to give operators and other systems a continuous overview of the systems performing the fast control loops.

CHAPTER 6

Contributions to appended papers

The work presented in this thesis was performed within the scope of two European projects; IMC-AESOP and Arrowhead. IMC-AESOP was a FP 7 project outlining an ArchitecturE for Service-Oriented Process (AESOP) - Monitoring and Control. The Arrowhead project was funded by the Artemis Joint Undertaking with a vision to enable collaborative automation by networked embedded devices.

The author has made contributions to different aspects of both of these projects, all of which focus on practical aspects of how to make the proposed solutions viable in real world situations within industrial automation and other similarly conservative and constrained environments.

This chapter presents an overview of the appended papers and highlights the contributions made by the author to each paper.

Paper A: Migration of industrial process control systems into service oriented architecture

Authors: Jerker Delsing, Fredrik Rosenqvist, Oscar Carlsson, Armando W. Colombo and Thomas Bangemann

Published in: Proceedings of 38th Annual Conference of the IEEE Industrial Electronics Society (IECON 2012), Montreal, Canada

This paper presents the basic outline of the migration approach suggested by the IMC-AESOP project, proposing four main migration steps consisting of 1) Initiation, 2) Configuration, 3) Data processing, and 4) Control execution.

Among the significant results presented in this paper is the analysis of interconnected industrial control systems and how subsystems may be disconnected from each other; the brief presentation of how the communication between the migrated and the legacy systems can be preserved using mediator technology, allowing continuous operation of the plant; and the discussion of how this can be applied in the case of an industrial DCS.

The author's contribution was to organize workshops with senior automation engineers and system designers and collect their input on how a Distributed Control System

could be described in terms of different characteristics and how these characteristics are usually interconnected. This allowed grouping them together into the four steps suggested by the paper, which through the discussion with the other authors could be formed into the more complete migration approach.

Paper B: Migration of a Legacy Plant Lubrication System to SOA

Authors: Philippe Nappey, Charbel El Kaed, Armando W. Colombo, Jens Eliasson, Andrey Kruglyak, Rumen Kyusakov, Christian Hübner, Thomas Bangemann, Oscar Carlsson

Published in: Proceedings of 39th Annual Conference of the IEEE Industrial Electronics Society (IECON 2013), Vienna, Austria

This paper reports the technical demonstration of service-oriented technology integrated in the control system of a live process plant. The paper shows how a combination of service-oriented devices from different partners, using different communication protocols can work together in a challenging environment.

This is an illustration of how a “cloud” can be initiated with many different protocols enabled in a very limited system, allowing easier expansion of the cloud at later migration stages or the addition of SOA systems to the migrated parts of the plant with the possibility to communicate with the legacy systems.

The author’s contribution to this paper was twofold; initially it was as an organizer of the demonstration and coordinator of integration tests and functional assessments for which plant owners had some concerns that had to be laid to rest before the new technology was allowed at the plant. Second, the author described how this demonstration of the Internet of Things technology in a real plant environment relates to a migration strategy.

Paper C: Migration of Industrial Process Control Systems into Service-Oriented Architectures

Authors: Oscar Carlsson, Jerker Delsing, Fredrik Arrigucci, Armando W. Colombo, Thomas Bangemann, Philippe Nappey

Submitted to: International Journal of Computer Integrated Manufacturing

This paper summarizes papers A and B, in addition to some material published in [61], and in the book produced by the IMC-AESOP consortium [69]. The paper presents the whole migration approach in detail and in one source as well as how it has been tested in the demonstration.

The author’s contribution was in the collection of results from the conference papers and book chapters, with significant editing and contextualizing the results with regards to related progress in the field. The author shows how all of these results together present a migration strategy that in the future can be used to allow a more widespread adoption of the Industrial Internet of Things, as it can be more easily applied to existing industrial process plants.

Paper D: Plant descriptions for engineering tool interoperability

Authors: Oscar Carlsson, Daniel Vera, Jerker Delsing, Bilal Ahmad, Robert Harrison

Published in: Proceedings of 14th IEEE International Conference on Industrial Informatics (INDIN 2016), Poitiers, France

This paper presents the Plant description concept, proposed as a part of the Arrowhead Framework for interoperability between IoT automation systems from various application domains. The paper highlights the high number of existing standards applicable to application domains covered by the Arrowhead project and presents the usability of the concept, both for engineering of IoT systems and in an automotive industry application.

The author's contribution was primarily in the collection and reflection on existing standards other related work, in detailing the Plant Description concept, in usage of the concept for engineering tool interoperability and in its usefulness for design and commissioning.

Paper E: Organizing IoT Systems-of-Systems from Standardized Engineering Data

Authors: Oscar Carlsson, Csaba Hegedűs, Jerker Delsing, Pal Varga

Published in: Proceedings of 42nd Annual Conference of the IEEE Industrial Electronics Society (IECON 2016), Florence, Italy

This paper extends the usability of the Plant description concept, first presented in Paper D, by detailing its applicability in Systems-of-Systems management and particularly how it can be used in connection with the Orchestration process for the Arrowhead Framework.

The author's contribution is mostly on the Plant description concept, the use case used for illustration of the process, and on compiling the contributions from the other co-authors.

Paper F: Configuration Service in Cloud based Automation Systems

Authors: Oscar Carlsson, Pablo Puñal Pereira, Jens Eliasson, Jerker Delsing, Bilal Ahmad, Robert Harrison, Ove Jansson

Published in: Proceedings of 42nd Annual Conference of the IEEE Industrial Electronics Society (IECON 2016), Florence, Italy

This paper presents several concepts, conceived during the Arrowhead project, that all contribute to more efficient deployment, configuration, and management of IoT devices. The concepts are illustrated with four use cases where the concepts can be combined to various degrees.

The author's contributions consist of the work with the Configuration store, the General deployment procedure, parts of the section "Intended areas of configuration",

and the use cases for Building automation and Process industries, as well as contributions to Related work, Discussion and Conclusions. The author also coordinated and edited most of the contributions from the other co-authors.

Paper G: Engineering of Industrial Internet of Things

Authors: Oscar Carlsson and Jerker Delsing

Submitted to: IEEE Systems Journal

This paper presents how many of the concepts introduced in Papers D, E, and F can be used together to form an engineering work-flow for IoT automation systems. The work-flow is illustrated through theoretical application in the engineering of a building heating systems, and compared to the engineering of traditional automation that may be used for this scenario.

The author's contributions consists of collecting and combining the concepts from the previously published material, forming the engineering work-flow, detailing the engineering of the scenario and the description of its application. The author was also responsible for the comparison with traditional automation, and most of the discussion and conclusions.

Paper H: The Arrowhead Framework architecture

Authors: Jerker Delsing, Pal Varga, Luis Ferreira, Michele Albano, Pablo Puñal Pereira, Jens Eliasson, Oscar Carlsson and Hasan Derhamy

Published in: IoT based Automation - made possible by Arrowhead Framework

This paper is the third chapter of the book produced by the Arrowhead consortium, describing the Arrowhead Framework and select applications. The paper describes the important principles of the framework and some key concepts and solutions.

The author's contribution was primarily in the design and description of the Automation support core systems PlantDescription and Configuration. Additional contributions include input to the discussions leading to some of the important definitions, e.g. distinction between Arrowhead Systems and Devices.

Paper I: Arrowhead Framework core systems and services

Authors: Jerker Delsing, Jens Eliasson, Michele Albano, Pal Varga, Luis Ferreira, Hasan Derhamy, Csaba Hegedűs, Pablo Puñal Pereira, and Oscar Carlsson

Published in: IoT based Automation - made possible by Arrowhead Framework

This paper is the fourth chapter of the book produced by the Arrowhead consortium, describing the Arrowhead Framework and select applications. The paper describes the details of the Arrowhead Framework core systems, and the services they provide.

The author's contribution was primarily in the design and description of the Automation support core systems PlantDescription and Configuration, and in a secondary role contributing to the discussions on other related core systems, most significantly the Orchestration system, the SystemRegistry and the DeviceRegistry.

Paper J: Engineering of IoT automation systems

Authors: Oscar Carlsson, Daniel Vera, Eduardo Arceredillo, Markus G. Tauber, Bilal Ahmad, Christoph Schmittner, Sandor Plosz, Thomas Ruprechter, Andreas Aldrian, and Jerker Delsing

Published in: IoT based Automation - made possible by Arrowhead Framework

This paper is the sixth chapter of the book produced by the Arrowhead consortium, describing the Arrowhead Framework and select applications. The paper describes the engineering of IoT automation systems, in the context of the Arrowhead Framework.

The author's contribution was in both the collection, coordination and editing of contributions from the other co-authors, as well as contributing with some sections. Among these, the author contributed significantly to the sections "Introduction", "Engineering of an Arrowhead compatible multidomain facility", and the engineering scenarios "Swift deployment and configuration", "Replacement of device", and "Device Configuration Upload".

6.1 Additional publications

This section lists publications containing contributions from the author's research work which are related to, but not included in this thesis.

1. Thomas Bangemann, Stamatis Karnouskos, Roberto Camp, Oscar Carlsson, Matthias Riedl, Stuart McLeod, Robert Harrison, Armando W. Colombo and Petr Stluka, "State of the Art in Industrial Automation", Book chapter, Industrial Cloud-Based Cyber-Physical Systems, Springer International Publishing Switzerland, 2014.
2. Jerker Delsing, Oscar Carlsson, Fredrik Arrigucci, Thomas Bangemann, Christian Hübner, Armando W. Colombo, Philippe Nappey, Bernard Bony, Stamatis Karnouskos, Johan Nessaether and Rumen Kyusakov, "Migration of SCADA/DCS Systems to the SOA Cloud", Book chapter, Industrial Cloud-Based Cyber-Physical Systems, Springer International Publishing Switzerland, 2014.
3. Philippe Nappey, Charbel El Kaed, Armando W. Colombo, Jens Eliasson, Andrey Kruglyak, Rumen Kyusakov, Christian Hübner, Thomas Bangemann and Oscar Carlsson, "Migration of a Legacy Plant Lubrication System to SOA", Book chapter, Industrial Cloud-Based Cyber-Physical Systems, Springer International Publishing Switzerland, 2014.

CHAPTER 7

Conclusions and Future Work

To conclude this thesis, some discussion on the presented work is warranted, with the intention to relate the work to the societal challenges and research questions presented in the Introduction.

The primary observation that initiated this research work was that many actors around the world predict a rapid increase in electronic sensors and other connected electronic devices. A second observation was that, for the current generation of automation systems, the engineering work is a substantial part of the cost for installation of new systems and that this cost is somewhat correlated with the number of connected sensors and other devices. From the combination of these two observations a number of questions arise, e.g., what kind of engineering work will be required to bring these predictions to fruition? Will these new system enable a more efficient approach to automation engineering than what is currently in use?

7.1 Conclusions

Through the qualitative discussions of the approaches presented in this these, primarily in the appended papers, there are significant indications that the engineering effort for installing automation systems can be lowered for IoT-based systems-of-systems, compared to the procedures currently in place for engineering and installation of most existing automation systems.

With regards to the three research questions initially stated, the following answers have been deduced through the work presented in the thesis:

Q1 *How can run-time and design-time knowledge efficiently be integrated to manage an automation system-of-systems?*

Using an abstraction of the design-time knowledge, accessible at run-time, in a format that is understandable for run-time system-of-systems management tools, the design-time knowledge can efficiently be used in the management of an automation system-of-systems. If the abstraction of design-time knowledge additionally can be

appended with run-time knowledge, then the combined abstraction of design-time and run-time knowledge becomes a powerful source of information for a system-of-systems management tool.

One such solution is the Arrowhead Framework core support systems, where the PlantDescriptor system and the Configuration system are examples of abstractions of a combination of design-time and run-time knowledge, accessible to system-of-systems management tools such as the Orchestration system.

Q2 *How can standardised engineering methodologies for capturing process and technical knowledge be utilised to manage a cross-domain IoT automation system-of-systems?*

Utilising some of the existing methods for organising automation components into a structure of objects and their relations to each other can allow the identification IoT automation systems and how they should interact with each other when organised into a system-of-systems.

One standard for capturing process and technical knowledge is IEC 62424 (CAEX), in which systems can be identified and the relations between systems can be abstracted. In Papers D, E and G it is shown how this can be abstracted into a data structure of nodes representing IoT automation systems and links between the nodes representing the interactions between these systems. As standards from other domains, such as the IEC 61850, also identify objects and connections between them, these can be added to the same structure of nodes and links allowing for the creation of links between nodes originating in different domains. All of these links can subsequently be used to orchestrate service connections managing a cross-domain IoT automation system-of-systems.

Q3 *What are the functional, performance and operational critical aspects of a successful migration of an industrial automation system to an IoT-based cloud solution?*

The literature studies in the field of migration strategies suggest that the success factors of a migration of industrial automation systems are much like many other large industrial projects in that they depend on a justified project plan, iterative execution of the planned steps, monitoring and control of the progress and subsequent updates to the project plan, until the requirements are met.

The experience from the technical demonstration of a migration of an industrial control system to an IoT-based cloud solution indicates that the basis for a successful migration is that all requirements relating to the system are identified and subsequently tested and verified against. Preferably such tests should be performed at all critical points of the migration process. This conclusion relates to functional, performance and operational requirements as well as those relating to organization and business.

In conclusion, this work has resulted in answers to the initial research questions. Furthermore, it can be argued that the procedures and methods presented in this thesis significantly contribute to solving some key challenges associated with the engineering of

IoT automation systems. Thereby, the work should help the adoption of many new technologies in European industries and in other domains of society, presumably improving the outlook for many current societal challenges.

7.2 Future work

There is still significant research and development work to be performed in order to verify the findings presented here in a wider scope, as well as to bring the presented solutions to a level where they can be employed practically to the benefit of industry and society.

For the Plant Description concept, there are several areas that should be further investigated. Within the Arrowhead Framework, it is described as a service provided by a system and that the information therein originates from standardised engineering data. In this alone, there are remaining uncertainties. One question is:

- How is the Plant Description to be populated and how is it to be maintained?

For the population, there is ongoing work using ontologies to create full semantic integration of engineering environments, e.g., by Moser and Biffl [70]. The solutions presented could be useful but will need some adjustment as the challenge for the Plant Description is different in that there is less information that needs to be captured, but it is intended to be used with more different standards involved. It is also possible that there are completely different approaches to extracting information from standardised engineering data into a Plant Description, this is something that could be investigated more.

An approach similar to the one presented in Papers E and G, using the Plant Description for populating the Orchestration system with data from engineering data, has been hypothesised to be efficient for populating an Authentication and Authorisation (AA) system. However, this has not been investigated, tested or evaluated at all. An interesting question in this area could be:

- Can the Plant Description be used to manage an Authentication and Authorisation system?

As it concerns management of basic system properties and their relation to other systems, there are a lot of similarities, but the requirements are somewhat different and raises new questions regarding trust between stakeholders compared to when the AA-system is managed manually.

Further developments of the Configuration system could include the generation of configuration files from engineering data. This could be studied both in a very specific case, intended to complement the approach for compatible devices, or in a more general study. E.g.

- Can changes to an AutomationML data set be compiled into an updated configuration for an IoT controller?
- For which standards for engineering data and configuration files can such compilation be expected to work?

The work presented here has been focused on the industrial domain, with only minor discussions on e.g. standards from other domains. Based on the interest gathered from other application verticals and how some of the adaptations of established methodologies are intended to improve dynamics of the systems, there should be benefits from using the principles in other domains as well. A more empirical study could investigate questions like:

- How does the approach proposed for IoT-based System-of-Systems automation engineering work in an infrastructure automation scenario, compared to existing solutions?

A different approach in this area would be:

- How can the applicability of the proposed approaches to other domains be investigated? How can it be tested and evaluated?

Both qualitative and quantitative results may be useful in these kinds of evaluations of the presented methods and strategies.

A somewhat different area of further investigations would be the changes that can be expected from further stakeholder integration, as discussed in both Chapter 2 and 4. Full integration of multiple stakeholders into a local automation cloud will raise several new questions regarding e.g. orchestration and configuration. New questions may include:

- What changes to the configuration of a device is the stakeholder in charge of the device allowed to make?
- How can changes to one system be regulated to limit the impact on stability and performance of connected systems from other stakeholders?
- Which stakeholders should be allowed to manage orchestrations between systems from different stakeholders?

Some of the questions that arise among interacting stakeholders should probably be settled in commercial agreements but it is quite likely that some of the agreements will require further control or traceability of e.g. the configuration of devices or design of orchestrations.

REFERENCES

- [1] J. Lee, B. Bagheri, and H.-A. Kao, “A cyber-physical systems architecture for industry 4.0-based manufacturing systems,” *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.
- [2] “Factories of the Future – Multi-annual roadmap for the contractual PPP under Horizon 2020,” EFFRA, Tech. Rep., 2013.
- [3] “European roadmap for industrial process automation,” ProcessIT.EU, Tech. Rep., 2013.
- [4] J. Pereira, European Commission, and Directorate-General for Communications Networks, Content and Technology, “Monitoring & control - Today’s market and its evolution till 2020 : final report of the study SMART 2007/047,” Tech. Rep., 2009.
- [5] L. Snyder, F. Baskett, M. Carroll, M. Coleman, D. Estrin, M. Furst, J. Hennessy, H. Kung, K. Maly, and B. Reid, *Academic Careers for Experimental Computer Scientists and Engineers*. The National Academies Press, 1994, no. 0-309-58568-6, ch. 1. What is experimental computer science and engineering?, pp. 9–33. [Online]. Available: http://www.nap.edu/openbook.php?record_id=2236
- [6] M. Maier, “Architecting principles for systems-of-systems.” *Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.
- [7] J. McCarthy, “Reminiscences on the history of time-sharing,” *IEEE Ann. Hist. Comput.*, vol. 14, no. 1, pp. 19–24, Jan. 1992. [Online]. Available: <http://dl.acm.org/citation.cfm?id=612400.612431>
- [8] Q. Hassan, “Demystifying cloud computing,” *CrossTalk – The Journal of Defense Software Engineering*, pp. 16–21, 2011.
- [9] E. Schmidt. (2006) Conversation with Eric Schmidt hosted by Danny Sullivan. [Online]. Available: <https://www.google.com/press/podium/ses2006.html>
- [10] Amazon. (2006) Announcing Amazon Elastic Compute Cloud (Amazon EC2) - beta. [Online]. Available: <https://aws.amazon.com/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2---beta/>

- [11] P. Mell and T. Grance, “The NIST definition of cloud computing,” Tech. Rep., 2011. [Online]. Available: <http://dx.doi.org/10.6028/nist.sp.800-145>
- [12] R. Cupek, A. Ziebinski, L. Huczala, and H. Erdogan, “Agent-based manufacturing execution systems for short-series production scheduling,” *Computers in Industry*, vol. 82, pp. 245 – 258, 2016. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0166361516301233>
- [13] H. Chang, A. Hari, S. Mukherjee, and T. V. Lakshman, “Bringing the cloud to the edge,” in *2014 IEEE Conference on Computer Communications Workshops (INFO-COM WKSHPS)*, April 2014, pp. 346–351.
- [14] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.
- [15] S. Yi, C. Li, and Q. Li, “A survey of fog computing: Concepts, applications and issues,” in *Proceedings of the 2015 Workshop on Mobile Big Data*, ser. Mobidata ’15. New York, NY, USA: ACM, 2015, pp. 37–42. [Online]. Available: <http://doi.acm.org.proxy.lib.ltu.se/10.1145/2757384.2757397>
- [16] L. L. Ferreira, M. Albano, and J. Delsing, “QoS-as-a-Service in the local cloud,” in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sept 2016, pp. 1–8.
- [17] J. Delsing, J. Eliasson, J. van Deventer, H. Derhamy, and P. Varga, “Enabling IoT automation using local clouds,” in *Proceedings World Forum - IoT 2016*. IEEE, Dec. 2016.
- [18] D. Evans. (2011) The Internet of Things [INFOGRAPHIC]. web. Cisco Internet Business Solutions Group. [Online]. Available: <http://blogs.cisco.com/diversity/the-internet-of-things-infographic>
- [19] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller, “A survey of commercial frameworks for the internet of things,” in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015, pp. 1–8.
- [20] B. Lydon, Ed., *Internet of Things: Industrial automation industry exploring and implementing IoT*, March / April 2014. [Online]. Available: <https://www.isa.org/standards-and-publications/isa-publications/intech-magazine/2014/mar-apr/cover-story-internet-of-things/>
- [21] Cisco, “White paper: Cisco VNI Forecast and Methodology, 2015-2020,” Cisco Visual Networking Index, Tech. Rep., 2016.
- [22] Cisco, “The Zettabyte Era – Trends and Analysis,” Cisco Visual Networking Index, Tech. Rep., 2016.

- [23] H. Derhamy, J. Eliasson, J. Delsing, P. P. Pereira, and P. Varga, “Translation error handling for multi-protocol SOA systems,” in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015, pp. 1–8.
- [24] H. Derhamy, “Towards Interoperable Industrial Internet of Things: An On-Demand Multi-Protocol Translator Service,” Licentiate thesis, Luleå University of Technology, 2016. [Online]. Available: <http://ltu.diva-portal.org/smash/record.jsf?pid=diva2:971142>
- [25] H. Derhamy, J. Eliasson, and J. Delsing, “IoT Interoperability - On-demand and low latency Transparent Multi-protocol Translator,” *IEEE Internet of Things Journal*, 2016, submitted.
- [26] J. Delsing, Ed., *IoT Automation - Arrowhead Framework*. CRC PRess, To appear Feb 2017 2016.
- [27] S. Mayer, E. Wilde, and F. Michahelles, “A connective fabric for bridging internet of things silos,” in *2015 5th International Conference on the Internet of Things (IOT)*, Oct 2015, pp. 148–154.
- [28] F. Moutinho, L. Paiva, P. Maló, and L. Gomes, “Semantic annotation of data in schemas to support data translations,” in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 5283–5288.
- [29] R. Mosshammer, A. Einfalt, A. Lugmaier, J. Hodges, and F. Michahelles, “Semantic annotation engine for smart grid applications,” in *2015 5th International Conference on the Internet of Things (IOT)*, Oct 2015, pp. 132–137.
- [30] (2016) Arrowhead framework wiki. [Online]. Available: https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page
- [31] E. Hayden, M. Assante, and T. Conway, “An Abbreviated History of Automation & Industrial Controls Systems and Cybersecurity,” SANS Institute, Whitepaper, aug 2014.
- [32] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, “Reconfigurable manufacturing systems: Key to future manufacturing,” *Journal of Intelligent Manufacturing*, vol. 11, no. 4, pp. 403–419, 2000. [Online]. Available: <http://dx.doi.org/10.1023/A:1008930403506>
- [33] F. Jovane and Y. Koren and C. R. Boér, “Present and future of flexible automation: Towards new paradigms,” *CIRP Annals - Manufacturing Technology*, vol. 52, no. 2, pp. 543–560, 2003.
- [34] *Programmable controllers - Part 3: Programming languages*, International Electrotechnical Commission IEC IEC 61131-3:2013, 2013.

- [35] V. Alyokhin, B. Elbel, M. Rothfelder, and A. Pretschner, “Coverage metrics for continuous function charts,” in *15th International Symposium on Software Reliability Engineering*, Nov 2004, pp. 257–268.
- [36] B. Vogel-Heuser, *Automation in the Wood and Paper Industry*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1015–1026. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-78831-7_58
- [37] IEC62264-3, *IEC 62264-3, Enterprise-Control System Integration - Part 3: Activity Models of Manufacturing Operations Management*, IEC Std., 2007.
- [38] W. Mann, *Practical Automation Specification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 797–808. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-78831-7_46
- [39] J. F. G. Fernández and A. C. Márquez, “International standards, best practices and maintenance management models as reference,” in *Springer Series in Reliability Engineering*. Springer Nature, 2012, pp. 33–59. [Online]. Available: https://doi.org/10.1007/978-1-4471-2757-4_2
- [40] P. Adolphs and U. Epple, “Status report: Rami4.0,” VDI/VDE-Gesellshact Mess- und Automatisierungstechnik, Tech. Rep., June 2015.
- [41] *IEC 81346 Industrial systems, installations and equipment and industrial products – Structuring principles and reference designations*, ISO/IEC Std.
- [42] M. Schleipen. (2014) Open standards for Industry 4.0 - Tools and offer around AutomationML and OPC UA. [Online]. Available: http://www.iosb.fraunhofer.de/servlet/is/46944/AutomationML_en.pdf?command=downloadContent&filename=AutomationML_en.pdf
- [43] A. T. Johnston, “OpenO&M and ISO 15926 Collaborative Deployment,” October 2009. [Online]. Available: <http://www.mimosa.org/presentations/openom-and-iso-15926-collaborative-deployment>
- [44] M. Göring and A. Fay, “Modeling change and structural dependencies of automation systems,” in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, Sept 2012, pp. 1–8.
- [45] O. Carlsson, D. Vera, J. Delsing, B. Ahmad, and R. Harrison, “Plant descriptions for engineering tool interoperability,” in *2016 14th IEEE International Conference on Industrial Informatics (INDIN)*, 2016.
- [46] O. Carlsson, C. Hegedűs, and P. V. Jerker Delsing, “Organizing IoT Systems-of-Systems from Standardized Engineering Data,” in *Industrial Electronics Society, IECON 2016 - 42nd Annual Conference of the IEEE*, 2016.

- [47] O. Carlsson, P. P. nal Pereira, J. Eliasson, J. Delsing, B. Ahmad, R. Harrison, and O. Jansson, “Configuration service in cloud based automation systems,” in *Industrial Electronics Society, IECON 2016 - 42nd Annual Conference of the IEEE*, 2016.
- [48] C. Hegedűs, D. Kozma, G. Soós, and P. Varga, “Enhancements of the Arrowhead Framework to refine inter-cloud service interactions,” in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 5259–5264.
- [49] IEC 62424 *Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools*, International Electrotechnical Commission Std.
- [50] R. Drath, A. Luder, J. Peschke, and L. Hundt, “AutomationML - The glue for seamless automation engineering,” in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, Sept 2008, pp. 616–623.
- [51] MIMOSA OSA-EAI - Open System Architecture for Enterprise Application Integration, MIMOSA OSA-EAI Std. [Online]. Available: <http://www.mimosa.org/mimosa-osa-eai>
- [52] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer, 2009.
- [53] V. Vyatkin, *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*, third edition ed. International Society of Automation, 2016.
- [54] fastAPI - Standard för fastighetskommunikation, SABO Std. [Online]. Available: <http://www.fastapi.se/>
- [55] IEC 61850: Power Utility Automation, International Electrotechnical Commission Std.
- [56] Building automation and control systems (BACS), ISO ISO 16 484-1:2010, 2010.
- [57] ISO/TS 18876 *Industrial automation systems and integration – Integration of industrial data for exchange, access and sharing*, International Organization for Standardization Std.
- [58] F. Blomstedt, L. Ferreira, M. Klisics, C. Chrysoulas, I. Martinez de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, “The arrowhead approach for soa application development and documentation,” in *Industrial Electronics Society, IECON 2014 - 40th Annual Conference of the IEEE*, Oct 2014, pp. 2631–2637.
- [59] J. Bisbal, D. Lawless, and R. Richardson, “A survey of research into legacy system migration,” Tech. Rep., 1997.
- [60] J. Bergey, D. Smith, and N. Weiderman, “DOD legacy system migration guidelines,” DTIC Document, Tech. Rep., 1999.

- [61] J. Delsing, J. Eliasson, R. Kyusakov, A. Colombo, F. Jammes, J. Nessaether, S. Karnouskos, and C. Diedrich, "A migration approach towards a SOA-based next generation process control and monitoring," in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, Nov 2011, pp. 4472–4477.
- [62] S. Karnouskos, A. W. Colombo, F. Jammes, J. Delsing, and T. Bangemann, "Towards an architecture for service-oriented process monitoring and control," in *36th Annual Conference of the IEEE Industrial Electronics Society (IECON-2010), Phoenix, AZ.*, 7–10 Nov 2010.
- [63] S. Karnouskos and A. Colombo, "Architecting the next generation of service-based scada/dcs system of systems," in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, Nov 2011, pp. 359–364.
- [64] A. W. Colombo, T. Bangemann, S. Karnouskos, J. Delsing, P. Stluka, R. Harrison, F. Jammes, and J. Lastra, Eds., *Industrial Cloud-based Cyber-Physical Systems: The IMC-AESOP Approach*. Springer, 2014, iISBN: 978-3-319-05623-4. [Online]. Available: <http://www.springer.com/engineering/production+engineering/book/978-3-319-05623-4>
- [65] L. O'Brien and D. Woll, "The control system migration survival manual," ARC Advisory Group, Tech. Rep., March 2010. [Online]. Available: <http://www.arcweb.com/featured-reports/The-Control-System-Migration-Survival-Manual.pdf>
- [66] S. Nabil and B. Mohamed, "Security ontology for semantic SCADA," in *CEUR Workshop Proceedings*, vol. 867, 2012, pp. 179–192. [Online]. Available: www.scopus.com
- [67] A. Gligor and T. Turc, "Development of a Service Oriented SCADA System," *Procedia Economics and Finance*, vol. 3, no. 0, pp. 256 – 261, 2012, international Conference Emerging Markets Queries in Finance and Business, Petru Maior University of Tîrgu-Mureş, ROMANIA, October 24th - 27th, 2012. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S2212567112001499>
- [68] S. Karnouskos, A. Colombo, T. Bangemann, K. Manninen, R. Camp, M. Tilly, P. Stluka, F. Jammes, J. Delsing, and J. Eliasson, "A SOA-based architecture for empowering future collaborative cloud-based industrial automation," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, Oct 2012, pp. 5766–5772.
- [69] A. 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*. Springer Publishing Company, Incorporated, 2014.
- [70] T. Moser and S. Biffl, "Semantic integration of software and systems engineering environments," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 1, pp. 38–50, Jan 2012.

Part II

PAPER A

Migration of industrial process control systems into service oriented architecture

Authors:

Jerker Delsing, Fredrik Rosenqvist, Oscar Carlsson, Armando W. Colombo and Thomas Bangemann

Reformatted version of paper originally published in:
Conference paper, IEEE IECON 2012, Montreal, 2012.

© 2012 IEEE. Reprinted, with permissions, from Jerker Delsing, Fredrik Rosenqvist, Armando W. Colombo and Thomas Bangemann, *Migration of industrial process control systems into service oriented architecture*, IEEE IECON, 2012.

Migration of Industrial Process Control Systems into Service Oriented Architecture

Jerker Delsing, Fredrik Rosenqvist, Oscar Carlsson, Armando W. Colombo and Thomas Bangemann

Abstract

The procedure of migrating SCADA and DCS functionality of the ISA-95 process automation architecture to a Service based automation architecture is discussed. Challenges in such migration are discussed and defined. From here the necessary migration technology and procedures are proposed. The critical migration technology is based on the mediator concept. The migration procedure is based on a functionality perspective and comprises four steps: initiation, configuration, data processing and control execution. It is argued that these steps are necessary for the successful migration of DCS and SCADA functionality in to the automation cloud.

1 Introduction

There is a demand in the process industry to modernize the process control equipment as aging process control systems become maintenance intensive; in addition older systems certainly lack technical capabilities and features of newer ones. Service Oriented Architecture (SOA) is seen as a promising candidate to support cross-layer integration to make distributed systems more interoperable. Such technology shift has been in progress at the enterprise system level for many years. The technology has been pioneered by IBM and others, important contributions have been made by many, some examples can be found in [1] e.g. This statement is the result of certain European collaborative projects, such as SIRENA [2], SODA [3], SOCRADES [4, 5, 6], that demonstrated the feasibility of embedding Web Services at the device level and integrating these devices with MES and ERP systems at upper levels of an enterprise architecture [5]. Other projects which indicate the same development of industrial control systems include the European Initiatives PLANTCockpit, <http://www.plantcockpit.eu/>; KAP, <http://www.kaproject.eu/>; ActionPlanT, <http://www.actionplant-project.eu/the-European-Embedded-Systems-Platform-Advanced-Research-&-Technology-for-EMbedded-Intelligence-and-Systems---ARTEMIS>, <http://www.artemis.eu/>.

Those projects listed here were mainly addressing factory automation, whereas the IMC-AESOP [7, 8, 9] project considers applying the SOA paradigm to approach the next generation of SCADA/DCS systems with major focus to the process industry and industrial process control systems. The SoA paradigm applied to Control and Automation

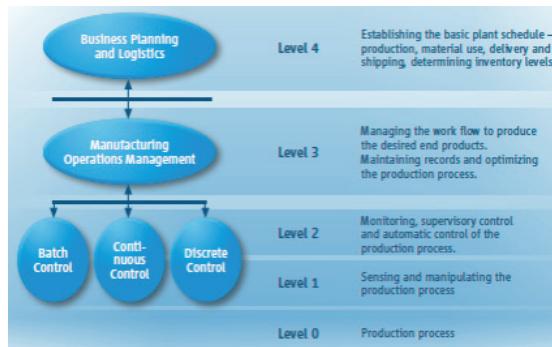


Figure 1: ISA95 architecture of automation system, functional hierarchy according to (IEC 62264-3) [11, 12]

provides with technologies, methods and tools that can enhance interoperability by decoupling functionality and their implementation. As a consequence, the transparency of the entire infrastructure, including systems development tools and devices, is increased.

Several provider of today's enterprise systems, Level 4 in the ISA-95 architecture please refer to Fig. 1, already support service-driven interaction e.g. via Web Services. Service Oriented Architecture is an approach used at this level. Services are also used for integration between Level 3 and Level 4 systems, available on the market. OPC UA [10] is a technology spreading-up to be used. PLCopen in close cooperation with OPC Foundation, defined a OPC UA Information Model for IEC 61131-3. A mapping of the IEC 61131-3 software model to the OPC UA information model, leading to a standard way how OPC UA server-based controllers expose data structures and function blocks to OPC UA clients like HMIs was defined [2]. OPC UA relies on Web Service based communication. Last year, a working group was established focussing on the definition of communication mechanisms via OPC UA for MES integration of Level 2 systems as well as the definition of the semantics for MES integration. Those activities can be seen as attempts to move towards the use of common technologies across different levels of production systems.

Legacy systems typically have proprietary protocols and interfaces resulting in vendor lock-ins and possibly site specific solutions; however with SOA these systems can be wrapped and integrated in a modern infrastructure. By abstracting from the actual underlying hardware and communication-driven interaction and focusing on the information available via services, the complete system is managed and controlled by service-driven interactions. Services can be dynamically discovered, combined and integrated in mash-up applications. By accessing the isolated information and making the relevant correlations, business services could evolve; acquire not only a detailed view of the interworking of their processes but also take real-time feedback from the real physical-domain services and flexibly interact with them.

The legacy systems are typically implemented following the 5-level model as defined

within the ISA 95 / IEC 62264 standard (<http://www.isa-95.com>). Operations, defined by that standard, are inherent to established production management systems [11]. In this context, concepts for integrating legacy systems, specifically on lower levels, into Service Oriented Architecture based systems can be seen as business enablers to take the customer from where she/he is today [9] into the future.

The novelty of migrating from a legacy process control system into a SOA, is to in a structured way, gradually upgrade highly integrated and vendor-locked standards into a more open structure while maintaining the functionality. This paper focuses on the process of migrating industrial process control systems into a SOA-based architecture. The challenges of step-wise migration of a highly integrated vendor-locked DCS (Distributed Control System) and/or SCADA (Super-visionary Control and Data Acquisition) are discussed. The approach taken addressing functionality. From here the necessary migration technology and procedures are proposed. The critical migration technology proposed is based on the mediator concept. The migration procedure proposed is based on a functionality perspective and comprises four steps: initiation, configuration, data processing and control execution. It is argued that these steps are necessary for the successful migration of DCS and SCADA functionality into a service-based automation cloud.

2 Challenges in migrating industrial process control systems

Today's control systems, as used in process or manufacturing automation, are typically structured in an hierarchical manner as illustrated in Fig. 1.

IEC 62264 (or originally ISA 95) [11] is the international standard for the integration of enterprise and control systems, developed to provide a model that end users, integrators and vendors can use when integrating new applications in the enterprise. The model helps to define boundaries between the different levels of a typical industrial enterprise. ISA 95/IEC 62264 define five levels. For each of these five levels certain problems and challenges becomes eminent when considering their implementation using a SOA based approach.

Whereas Level 0 is dedicated to the process to be controlled itself, Level 1 connects the control systems to the process by sensors and actuators. Through the sensors the control system can receive information about the process and then regulating the process through the actuators. Sensors convert temperature, pressure, speed, position etc. into either digital or analogue signals. The opposite is done by actuators. Including not only valves but also motors and motor equipment such as frequency converters in actuators, it can be said that the level of installed intelligence varies very much. Legacy implementations use a scan based approach reading and writing data from/to sensors/actuators. Which differs fundamentally to the event based nature of a SOA approach [8, 13]. Migration on Level 1 has to some extent been described by Delsing et. al. [14] with focus on transition from scan-based to SOA event based communication when it comes to analogue signals.

At Level 3, operational management of the production is done, where Manufactur-

ing Execution Systems (MES) provide multiple information and production management capabilities. In the context of control hierarchy, however, its main function is the plant-wide production planning and scheduling. In a continuous process plant, the results of scheduling are used as production targets for individual shifts, and consequently, translated by engineers and operators into individual set points and limits. Level 3 integrates information about production and plant economics and provides detailed overview about the plant performance. If the production is straight forward with few articles and small production site, a dedicated Level 3 system might not bring added value. Some typical MES/MIS functionality is instead put in Level 2 and/or in the ERP-system (Level 4). At Level 4, typically Enterprise Resource Planning Systems (ERP) are installed for strategic planning of the overall plant operation according to business targets. Migration into SOA at Level 3 and 4 does not differ significantly for factory automation and process control systems [6]

At Level 2 there are some non-resolved challenges of migration when it comes to the process industry. Distributed monitoring and control enables plant supervisory control. The distributed control system (DCS) of a large process plant is usually highly integrated compared with a SCADA solution which is standard in factory automation. The SCADA is a supervisory system for HMI and data acquisition and the system communicates through open standard protocols with subordinated PLCs. The PLCs in the SCADA solution are autonomous compared to their counterpart, which sometimes are referred to as controllers, in the DCS. In this paper the process control system is defined as a DCS including HMI workstations, controllers, engineering station and servers all linked by a network infrastructure. A DCS is truly "distributed," with various tasks being carried out in widely dispersed devices. Migration of Level 2 functionality in the form of a DCS exhibits challenges when it comes to co-habitation between legacy and SOA as well as the migration of the control execution [8, 13]. In this paper the DCS is exemplified by a server/client based system as depicted in Fig. 2, which is a common topology.

When migrating the DCS into SOA there are certain requirements based on expectations from business, technical and personnel perspectives:

- The new architecture and the migration strategy must assure the same level of reliability and availability as the legacy system.
- The migration procedure must not induce any increased risk for staff, equipment or process reliability and availability.
- After the migration the plant must still provide the same or a better process, extended service life of plant (process equipment e.g. pumps, vessels, valves), adequate information and alarms depending on department and personnel skill and improved vertical (cross-layer) communication with more information available at plant wide level.
- Dynamic changes and reorganization is expected to be supported, on a continuously running system.

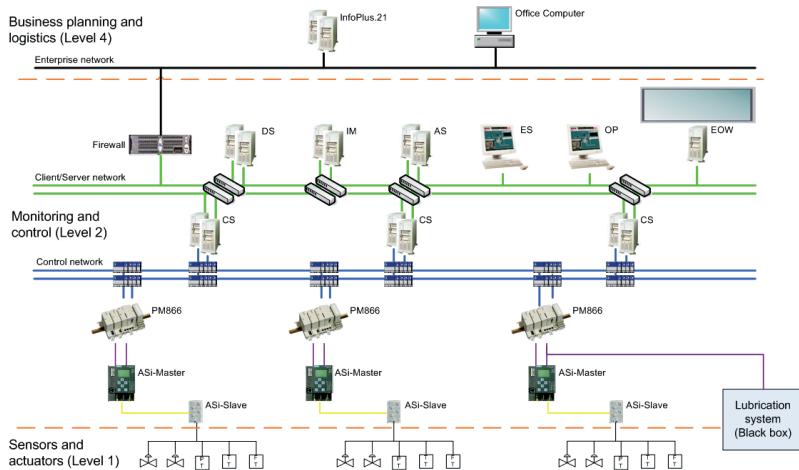


Figure 2: Legacy system architecture

- To handle co-habitation between the legacy system and the SOA during the migration phase, the SOA solution must support wrapping of legacy sub systems.
- Fieldbus systems, like Profibus PA today already define standardized ways of error indication by devices [15]. With the intelligence built into SOA devices troubleshooting is expected to be improved.

In order to migrate a highly integrated DCS the following challenges should be addressed:

- **Preserve functional integration:** There are advantages with a highly integrated DCS, which give a tight link between the HMI and control execution. Thus design engineering, commissioning and operation can be pursued in a significantly more uniform way. For instance, the HMI and control execution can be configured by the same tools, which facilitates conformity. These advantages must be maintained even though the integration is broken down and substituted by open standards.
- **Grouping of devices:** Within a given system, it must be determined which devices should be migrated to SOA as devices and which devices should be grouped together and the group migrated to SOA. As example a subsystem using feedback and regulation might require legacy interfaces because of real-time demands, therefore such group of devices should be given an SOA interface for the group using a Mediator and not at device level. This part of the system may be handled as "black-box"
- **Preserve real-time control:** The real time control execution, which in the legacy system is secured in the controllers, must be preserved.

3 Migration procedure

Interfacing and integrating legacy and SOA components of a DCS/SCADA system will require some, for the purpose developed and/or adapted, technology. Such integration may be based on some kind of integration component like Gateway or Mediator. Such Gateway or Mediator have the task to bridge the communication from major standardized protocols used close to field applications today: HART communication supported by HCF, Profibus PA in combination with Profibus DP, Foundation Fieldbus, etc. These protocols follow specific characteristics. Some commonalities can be monitored like concepts for device descriptions or integration mechanisms into DCS (e.g. EDD, FDT, FDI). The same bridging task exist regarding communication to higher level, technologies related to Enterprise Application Integration (EAI) or Enterprise Service Bus (ESB) or OPC (OPC DA, OPC UA) are used having their own characteristics and configuration rules.

The use of Gateways or Mediators is a well proven concept for integrating/connecting and migrating devices, attached to different networks. It is used to transform protocols as well as syntax of data. Semantic integration is hard to achieve. Nevertheless it is possible to do transformation between data centric approaches, as typically followed by fieldbus concepts, and service oriented, event centric, approaches.

The Mediator [16] concept used here is built on the basis of the Gateway concept by adding additional functionality. Originally meant to aggregate various data sources (e.g. databases, log files, etc.), the Mediators components evolved with the advent of Enterprise Service Buses (ESBs) [17]. Now a Mediator is used to aggregate various non WS-enabled devices or even services in SOAs. Using Mediators instead of a Gateways, provides the advantage of introducing some semantics or to do pre-processing of data coming from legacy networks, e.g. representing a package unit. Due to the diversity of data, or different aspects of interest, that different applications request different types (e.g. quality, quantity and granularity) of data, interface devices will normally be built as a combination of Gateway and Mediator. As it may also be applicable to integrate service oriented sections (e.g. retro-fit of a plant section or replacement of a package unit) into existing systems, this Gateway and Mediator concept can be extended to represent services into data centric systems (today's legacy systems). Mediator as well as Gateway concepts, both are powerful means for integrating single legacy devices or legacy systems encapsulating "isolated" functionalities. Whereas the operational phase of a system will benefit from the functionalities described above from the beginning of the migration process, engineering will be characterized by a step-wise approach, starting with defining services representing the legacy device or system, followed by separate engineering steps for the legacy part and the SOA based part using those services defined. Specific configuration effort for the Mediator or Gateway itself is needed. It is advisable, that commissioning will also be done in a multiple step approach, starting at the isolated components followed by their integration into the overall system.

Considering the layout of a server/client-based SCADA/DCS a stepwise migration through four major steps is proposed. The four major steps may contain sub-steps and

may be spread out over a long period of time but each major step should be completed before the following step is initiated. The four major steps suggested are:

- Initiation
- Configuration
- Data processing
- Control execution

During the whole migration the system will require one or more mediators to allow communication between the SOA components and the parts of the legacy system that not yet has been migrated. The propagation of the mediator and the growth of the SOA cloud are exemplary applied to the migration of the legacy SCADA/DCS presented in Fig. 2. Making emphasis in the DCS-part, the set of Fig.s 3 to 6 shows the different results reached throughout the whole migration process.

3.1 Step 1: Initiation

The initial SOA “cloud” needs some of the basic services presented in [18] in order to support basic communication and management of the cloud. Once the basic architecture is constructed the first peripheral subsystems can be migrated and new components can be integrated in SOA. In migration of subsystems, as well as integration of new components, some consideration must be made of the limitations of the mediator and its communication paths.

The systems migrated in this step include sub system which are not directly part of the highly integrated DCS:

- Low level black box
- High level systems for business planning and logistics such as maintenance systems

Migration is limited to the operational phase of the systems integrated. Within that step, engineering is out of the scope of migration. An appropriate engineering approach, dedicated to this migration step, is doing multi-step configuration:

- Configuration of every legacy system including the legacy interface within the mediator
- Configuring the SOA system
- Configuring the model mapping within the mediator

Exploiting machine readable legacy configuration information would be helpfull for every step. Today, configuration information is available through different technologies e.g. GSD, DD, paper documentation. This type of information is mostly available for single devices. Engineering stations take these information as input and generate system configuration information in proprietary formats.

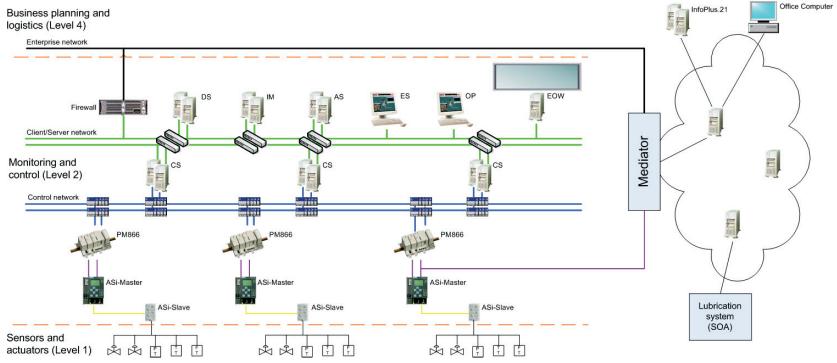


Figure 3: DCS after the first step of migration

3.2 Step 2: Configuration

This is the first step where components that are heavily integrated in the DCS are migrated. The purpose of this step is to migrate parts of the DCS that do not require very short response times or the regular transport of large amounts of data. Please refer to Fig. 4. The majority of functions that qualify for this migration step are in some way concerned with configuration of different parts of the DCS. The point of origin for most, if not all, configuration is the Engineering Stations (ES) which is used for engineering and configuration of most parts of the DCS.

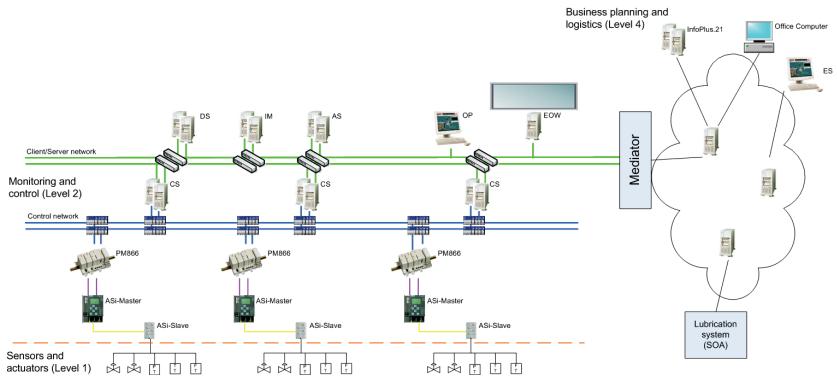


Figure 4: DCS after the second step of migration

As the ES is migrated to SOA, this constitutes a major increase in the number of services the Mediator needs to supply to the SOA cloud as it must in addition to the operational data migrated in the first step represent configuration aspects of all

legacy systems and devices not yet migrated, and allow configuration of all systems and devices. This means that configuration of low-level devices and control is done on the ES in a SOA environment using configuration services provided by the mediator, the configuration is then compiled by the mediator into their respective legacy formats and downloaded into the legacy controllers. Configuration of HMI, Faceplates and associated systems is similarly done in SOA and converted by the mediator to a format that can be downloaded into the legacy Aspect servers and other legacy systems. The configuration of legacy devices from SOA might also require that the mediator is able to extract legacy designs and configurations that may be stored in aspect servers or controllers so that old designs and be reused and modified by the SOA Engineering stations.

As legacy systems usually do not provide sufficient meta-data, sufficient configuration information can not necessarily be extracted by a Mediator from the installation (legacy systems). Today, configuration information is available through different technologies (GSD, DD, paper documentation, ...). This type of information is mostly available for single devices. Engineering stations take these information as input and generate system configuration information in proprietary formats not necessarily interpretable by other tools.

Consequently, for overall engineering a SOA engineering station should be able to import relevant configuration information of different legacy systems in addition to the limited capabilities provided by the Mediator itself. If such a tool would be available, one could design a mediator acting as configuration station for different legacy systems (compile configuration information into legacy formats) while receiving basic configuration information from the SOA engineering station.

This approach may be combined with doing multi-step configuration described in the former step.

3.3 Step 3: Data processing

In this third step, the migration includes all components and/or subsystems that do not require short response time (millisecond range) not currently achievable by the SOA technology. Please refer to Fig. 5. This includes Operator Clients (OP) and Operator Overview Clients (EOW) as well as Aspect Servers (AS) and Information Management Servers (IM). As all points of user interaction with the system is now moved to SOA this means that the legacy Domain Servers (DS) are redundant. However, as user management and security needs to be available in SOA from the first step of the migration, there is probably no need for the Domain Servers in the SOA cloud, although the functionality can be considered to be migrated.

The migration the Operator Clients and the Aspect and Information Management servers mean that the role of the mediator is once again fundamentally changed. In Step 3 of the migration there is less of a need for a flexible mediator that can communicate with a lot of different legacy components, the new requirements are more concerned with a need to present large amounts of data available from legacy controllers to the migrated Operator clients and other data processors and consumers. This activity is closely related

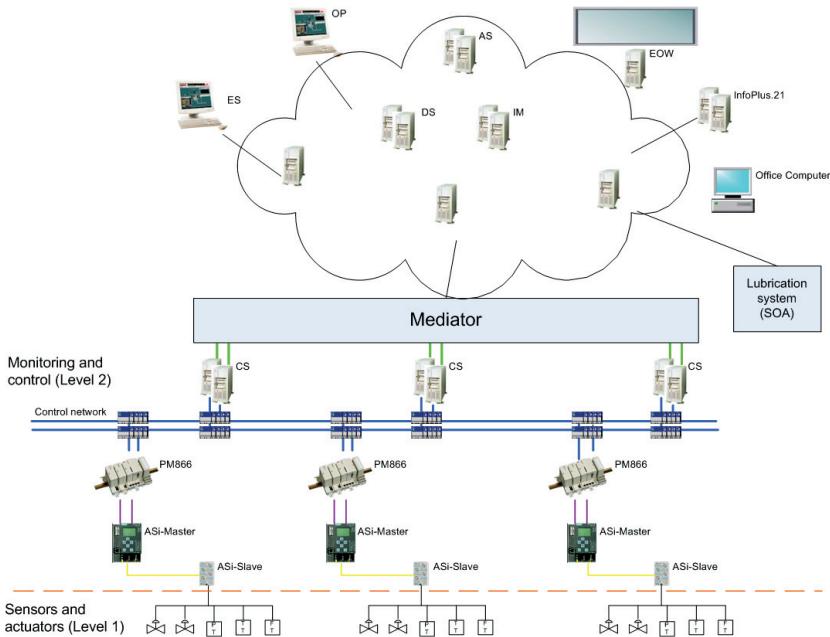


Figure 5: DCS after the third step of migration

to the purpose of the Connectivity Servers (CS) and it is suggested that the mediator in Step 3 is implemented as a new interface in the Connectivity Servers.

3.4 Step 4: Control execution

In the fourth and final step of migration the time has come to migrate the functionality traditionally provided by controllers. Please refer to Fig. 6. As control execution in the legacy system can be grouped together with several control functions in one controller, or in some cases spread out with different parts of a control function executed by more than one controller, it is of outmost importance that control execution is migrated function by function rather than controller by controller.

Depending on the performance requirements of each control function there may be a need for different strategies for different functions. In the cases where SOA compliant hardware is available for all functions an Active Migration may be suitable where a detailed schedule can be made over the migration of all functions, enabling a controlled migration towards a set deadline. In other cases it may be suitable to allow legacy controllers to fade out as functions are migrated in the course of normal maintenance and lifecycle management of the plant. The fade out option means that Step 4 of the migration may take a very long time but it may save costs as legacy devices are used for their full lifetime, while most benefits of SOA are already available.

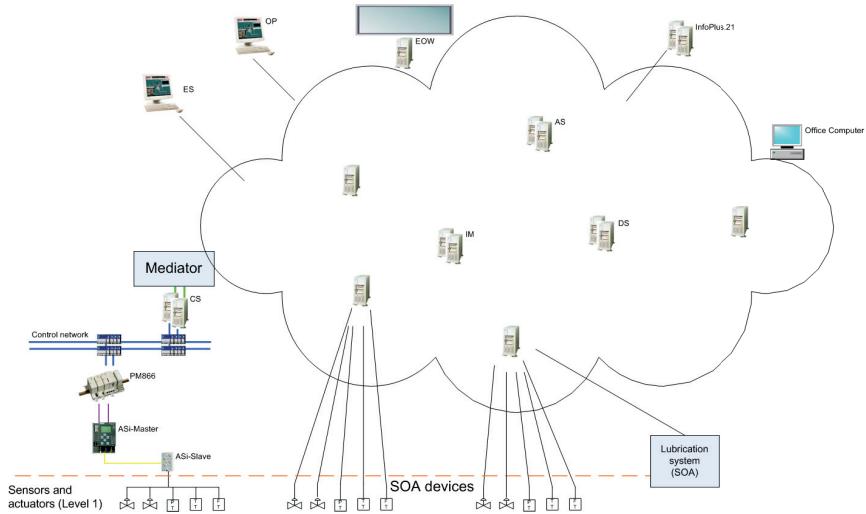


Figure 6: DCS after the forth step of migration

4 Conclusion

Following and extending the initial migration concepts introduced in [14], the novelty of migrating from a ISA'95-based legacy process control system into a SOA is to proceed in a structured way, gradually upgrading highly integrated and vendor-locked standards into a more open structure while maintaining the functionality. A procedure migrating the functionality of a DCS/SCADA to a cloud SOA based implementation is proposed. The procedure comprises 4 distinct steps and make use of mediator technology. These 4 steps are designed to maintaining the feeling of conformity between HMI and control execution and that the target system must exhibit full transparency and support open standards. It is important that the initiation in Step 1 consider these issues in order to enhance the plug-and-play feature of a SOA system even in an industrial process control system. The parts of the DCS/SCADA where operations and engineering are handled will in a structured way be migrated in Step 2 and Step 3. When it comes to the control execution, its migration approach is decided upon based on functionality and real-time requirements. If utilizing the fade out approach the most critical control loops and control logics may stay with their legacy set-up, in which operational, engineering and maintenance staff are confident. When these critical functions finally are upgraded they will be completely SOA compatible, whereas the real-time execution is run on device level.

Using this step vise approach, utilizing SOA and mediator technology, its is argued that the SOA approach will: preserve functional integration, support grouping of devices, preserve real-time control and successful addressing of safety loops. Making emphasis in the DCS-part of an exemplary legacy SCADA/DCS, the authors applied the approach

and present the results reached throughout the whole migration process.

Within the scope of IMC-AESOP project [7] the next step aims at evaluating the results of applying the migration procedure in dedicated industrial use-cases. Having in mind that one of the major results of the migration is the transformation of the ISA-95 architecture into a Automation Service Cloud. Future work will be oriented to extend the migration process and procedures to comprise the full ISA-95 architecture including security issues.

Acknowledgment

The authors would like to thank for their support the European Commission, and the partners of the EU FP7 project IMC-AESOP (www.imc-aesop.eu) for the fruitful discussions.

References

- [1] T. Erl, *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. The Prentice Hal, 2005.
- [2] H. Bohn, A. Bobek, and F. Golatowski, “Sirena - service infrastructure for real-time embedded networked devices: A service oriented framework for different domains,” in *Proc. of the International Conference on Networking, Systems, Mobile Communications and Learning Technologies*, IEEE Computer Society. IEEE Computer Society, 2006.
- [3] S. de Deugd, R. Carroll, K. E. Kelly, B. Millett, and J. Ricker, “Soda: Service oriented device architecture,” *Pervasive Computing, IEEE*, vol. 5, no. 3, pp. 94–96, july-sept. 2006.
- [4] A. W. Colombo, S. Karnouskos, and J. M. Mendes, “Factory of the Future: A Service-Oriented System of Modular, Dynamic Reconfigurable and Collaborative Systems,” in *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management*, L. Benyoucef and B. Grabot, Eds. Springer, 2010.
- [5] S. Karnouskos, D. Savio, P. Spiess, D. Guinard, V. Trifa, and O. Baecker, “Real World Service Interaction with Enterprise Systems in Dynamic Manufacturing Environments,” in *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management*. Springer, 2010.
- [6] A. W. Colombo and S. Karnouskos, “Towards the factory of the future: A service-oriented cross-layer infrastructure,” in *ICT Shaping the World: A Scientific View*. European Telecommunications Standards Institute (ETSI), John Wiley and Sons, 2009, vol. 65-81.

-
- [7] (2012) Aesop - architecture for service oriented process monitoring and control. [Online]. Available: <http://www.imc-aesop.eu>
 - [8] S. Karnouskos, A. W. Colombo, F. Jammes, J. Delsing, and T. Bangemann, “Towards an architecture for service-oriented process monitoring and control,” in *Proceedings IECON 2011*. IEEE Industrial Electronics Society, 2011, p. 6.
 - [9] S. Karnouskos and A. W. Colombo, “Architecting the next generation of service-based scada/dcs system of systems,” in *Proceedings IECON 2011*, Melbourne, Nov. 2011, p. 6.
 - [10] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer, 2009.
 - [11] “Iec 62264-3, enterprise-control system integration - part 3: Activity models of manufacturing operations management,” IEC, Tech. Rep., 2007.
 - [12] “Manufacturing execution systems (mes): Industry specific requirements and solutions,” <http://www.zvei.org/automation/mes>, ZVEI, Tech. Rep. ISBN: 978-3-939265-23-8, 2011.
 - [13] F. Jammes and H. Smit, “Service-oriented paradigms in industrial automation,” *Industrial Informatics, IEEE Transactions on*, vol. 1, no. 1, pp. 62 – 70, feb. 2005.
 - [14] J. Delsing, J. Eliasson, R. Kyusakov, A. Colombo, F. Jammes, J. Nessaether, S. Karnouskos, and C. Diedrich, “A migration approach towards a soa-based next generation process control and monitoring,” in *IECON 2011*, ser. Annual Conference of the IEEE Industrial Electronics Society, 2011, pp. 4472 – 4477.
 - [15] C. Diedrich and T. Bangemann, *PROFIBUS PA Instrumentation Technology for the Process Industry*. Oldenbourg Industrieverlag GmbH, 2007, no. ISBN-13 978-3-8356-3125-0.
 - [16] (2012) Socrades: A web service based shop floor integration infrastructure. [Online]. Available: <http://www.socrades.eu/Home/default.html>
 - [17] C. Héault, G. Thomas, and P. Lalanda, ““mediation and enterprise service bus: a position paper” in first international workshop on mediation in semantic web services,” in *Proc. MEDiate 2005*, 2005.
 - [18] S. Karnouskos, A. W. Colombo, K. Manninen, R. Camp, M. Tilly, T. Bangemann, P. Stluka, F. Jammes, J. Delsing, and J. Eliasson, “A soa-based architecture for empowering future collaborative cloud-based industrial automation,” in *Proc IEEE IECON 2012*. Montreal, Canada: IEEE, Oct. 2012.

PAPER B

Migration of a Legacy Plant Lubrication System to SOA

Authors:

Philippe Nappey, Charbel El Kaed, Armando W. Colombo, Jens Eliasson, Andrey Kruglyak, Rumen Kyusakov, Christian Hübner, Thomas Bangemann and Oscar Carlsson

Reformatted version of paper originally published in:

Conference paper, IEEE IECON 2013, Vienna, 2013.

© 2013 IEEE. Reprinted, with permissions, from Philippe Nappey, Charbel El Kaed, Armando W. Colombo, Jens Eliasson, Andrey Kruglyak, Rumen Kyusakov, Christian Hübner and Thomas Bangemann, *Migration of a Legacy Plant Lubrication System to SOA*, IEEE IECON, 2013.

Migration of a Legacy Plant Lubrication System to SOA

Philippe Nappey, Charbel El Kaed, Armando W. Colombo – Schneider Electric

{philippe.nappey, charbel.el-kaed, armando.colombo}@schneider-electric.com

Jens Eliasson, Andrey Kruglyak, Rumen Kyusakov – Luleå University of Technology, Sweden

{jens.eliasson, andrey.kruglyak, rumen.kyusakov}@ltu.se

Christian Hübner, Thomas Bangemann – ifak e.V., Germany

{christian.huebner, thomas.bangemann}@ifak.eu

Oscar Carlsson – Midroc, Sweden

oscar.carlsson@midroc.se

Supervision and control systems are being deployed in industrial environments such as mining plants and manufacturing facilities to ensure a continuous and effective production at a minimum cost. Such systems monitor a whole range of devices and collect their data for several purposes like maintenance and control operations.

At the same time, Service Oriented Architecture (SOA) is getting more popular than ever in most application domains, from IT to device level, and particularly for those industries whose continuous efforts to increase the overall plant and equipment effectiveness lead to new requirements on systems openness, integration, availability, maintainability and performance.

This paper presents how a legacy control and monitoring system could migrate to SOA, the architecture principles and the main benefits and limitations based on results validated on a pelletizing plant lubrication system.

I. INTRODUCTION

The IMC-AESOP project has been investigating for the past two years how a Service Oriented Architecture (SOA) would benefit to large scale distributed systems in Batch and Process Control applications. The project addresses in particular architectures where large number of service-compliant devices and systems distributed across a whole plant-wide system expose SCADA/DCS monitoring and control functions as Services. More detailed description of the project motivation and high level architecture of SOA enabled process monitoring and control is presented by Karnouskos et al. [11].

One essential investigated aspect has been the co-habitat of currently used synchronous DCS and SCADA with the new asynchronous SOA-based monitoring and control system, going beyond what the current implemented control and monitoring systems are delivering today. We'll detail in this paper the development of an IMC-AESOP demonstrator at the premises of LKAB in Sweden (see Figure 1), implementing an overall control scenario for an existing plant lubrication system and addressing the migration aspects between classical control systems and the new approaches addressed by the project.

Lubrication systems are typical critical systems for almost all process industries. The lubrication control system provides important information that can be used by operators to avoid critical and damaging incidents, by operators, planning staff and management to improve production and plant efficiency

and by the maintenance staff and management to analyze and improve the predictive maintenance.



Figure 1: LKAB Plant for Fe-Mineral Processing

The IMC-AESOP plant lubrication use case addresses a number of key points, such as enabling SOA on low level devices, SOA in closed-loop control, integration into an actual plant environment and migration from a scan based PLC to an event based SOA system. In [11], Karnouskos et al. discussed the advantages of using SOA-based solutions for industrial process monitoring and control.

In the following sections, we first outline the existing control and monitoring system then we describe the proposed IMC-AESOP architecture and components. Section IV summarizes the migration aspects into an SOA based solution while section V depicts the main implementation choices. Finally, section VI provides the outcomes of the validation in the real plant facility.

II. PROTOTYPE ARCHITECTURE

The lubrication system, presented in Figure 2, is deployed in the LKAB pelletizing plant [13] on a number of independent black-box systems. Such systems have limited data exchanges between the lubrication systems and the larger distributed control system (DCS). One of these black-box systems will be migrated from the current implementation using a PLC to a SOA system.

Similar migration efforts are described in the work of Feldhorst et al. in [17] where they use XML/DPWS exclusively as a SOA implementation. In order to extend the service approach to highly constrained embedded devices we propose to use binary encoding for XML and the application protocols which is not investigate in the aforementioned work.

A. Existing system

As shown in Figure 2, the existing lubrication system includes two lubrication circuits controlled by a PLC (Programmable Logic Controller) receiving start/stop commands from a DCS. Each lubrication circuit is connected to a pump controlled by the DCS through a digital output. More than 70 AS-i [14] position switches combined with various digital inputs are scanned periodically by the PLC to get fluid distribution status over each lubrication circuit. Based on this sensors information the PLC controls each pump and directs the fluid to the appropriate circuit. As mentioned above, there is a very limited communication with operational layer, although a touch panel provides a local supervision capability.

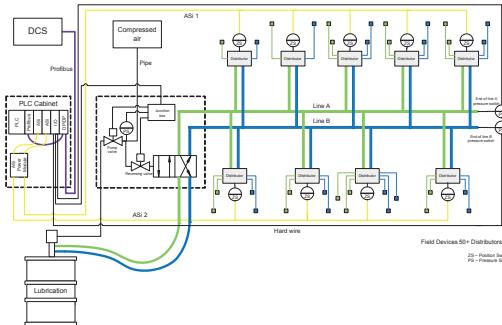


Figure 2: Existing system

B. Proposed prototype

The prototype proposed for IMC-AESOP consists in replacing the existing PLC with an SOA-based system. Thus, the current PLC cabinet is replaced with a SOA-based cabinet and connected to a maintenance station (SCADA), as shown in Figure 3.

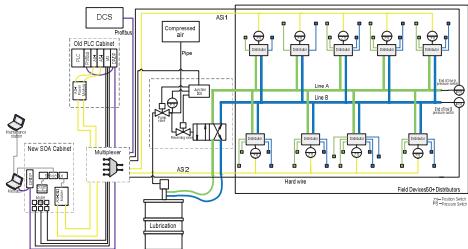


Figure 3: Proposed prototype

III. SOA COMPONENTS

The proposed SOA architecture is illustrated in Figure 4.

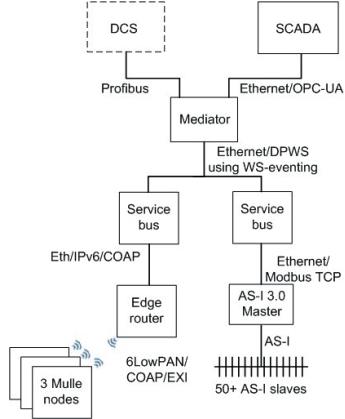


Figure 4: Proposed prototype SOA architecture

A. Mediator

The Mediator provides a runtime system for monitoring and control of process facilities by integrating both legacy as well as SOA-based technologies. It has been built based on an actor-based middleware for fault-tolerant, distributed SCADA systems. The adoption of the actor model [3] for the Mediator implementation results in less complexity and increased reliability compared to conventional (thread-based) approaches to the programming of concurrent processes. As all relevant subsystems are actors that interact with each other only by message passing without sharing common data structures, the actor-based design of the Mediator also greatly simplifies the distribution of parts of the Mediator system. Figure 5 shows the basic structure of the Mediator. The core part of it consists of a data model that describes the logical view of the monitored facilities and also contains all relevant information for acquiring data including communication. The Mediator communicates with the Service Bus through DPWS and also supports basic authentication over SOAP.

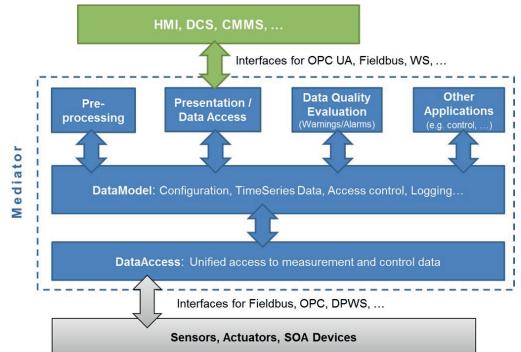


Figure 5: The Mediator structure

For the integration of different communication protocols and information models of various devices and other data sources, an abstract data access layer has been introduced. By providing adapters implemented as actors, any required protocol can be integrated. For the application described in this paper, the PROFIBUS protocol (for connecting to the DCS) as well as the DPWS protocol (for connecting to the Service Bus) has been implemented. In a similar fashion, any processing of the data for pre-processing, control, KPI calculation or presentation to the SCADA HMI layer is easily extendable by providing appropriate adapters.

Within the framework of this SOA system described above, the Mediator data model (including alarms) is presented to the HMI of the maintenance application using the OPC UA protocol.

B. Distributed Service Bus (DSB)

In complement to the Mediator, the Distributed Service Bus provides an additional integration of heterogeneous systems supporting various communication media, protocols, and data models, as shown in Figure 6. Such integration is enabled through loose coupling-based protocol connectors. Each protocol connector reifies devices and services of an existing system into the DSB data model representation. Thus, the Service Bus provides, through a defined abstract layer, a common representation of those devices and services. This abstract layer enables a wide variety of common operations on the underlying systems and devices, including management, diagnostic, maintenance and monitoring.

Moreover, the distribution of the Service Bus provides scalability and evolvability, as each instance can be configured for a specific application domain by implementing dedicated interfaces, quality of service and security requirements. Devices and services handled by an instance of the Service Bus are reified and their information is shared between the other instances through the DPWS protocol.

The distribution feature provided by the DSB is particularly suited to the management of large scale distributed systems, which is central to IMC-AESOP and this demonstrator in particular.

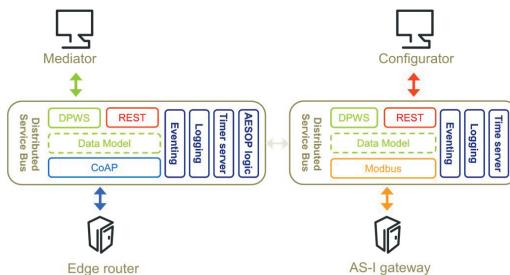


Figure 6: The DSB architecture

C. Industrial Internet of Things

The recent use of internet protocols and web technologies for distributed sensor network installations is gaining wider acceptance [16]. The wireless sensor and actuator network

(WSAN), i.e. an industrial approach to Internet of Things (IoT) [10], is built on the 868MHz version of the IEEE 802.15.4 radio standard, which enables low-power communication through thick concrete walls and long-range communication at line of sight operation. The use of Industrial Internet of Things (IIoT) is suitable to use in combination with lightweight embedded systems that are used to measure (and control) the physical parameters of interest.

To make the system scalable and integrate with the IMC-AESOP service cloud, IPv6 was chosen as network protocol. To make the IPv6 network layer comply with the IEEE 802.15.4 Link layer, the 6LoWPAN adaptation layer is used, 6LoWPAN compresses and reduces the data overhead so less energy is required to transfer the information between wireless nodes. IPv6 also enables unique identification of every sensor node using 128bit IPv6 address. The use of IPv6 also by default includes the network layer security feature of IPsec. Figure 4 shows the edge router which performs translation between IPv4 over Ethernet and IPv6 over 6LoWPAN (IEEE 802.15.4) networks. The edge router also hosts time synchronization services (NTP and PTP) and CoAP services such as data proxy, and also logs the performance of the WSAN. CoAP is a protocol designed for scalability and simplicity [20], whilst being backwards compatible with the much used HTTP protocol.

Mulle [12] devices serve as I/O nodes connecting lubrication pressure switches, air pressure switches, pump valves, reversing valves, and indication lights. Mulle nodes communicate using Efficient XML Interchange (EXI) [4] and CoAP on top of 6LoWPAN. The services hosted by the Mules supports input, output, filtering, logging, and configuration services. All the data are EXI encoded and transmitted using CoAP over 6LowPAN.

Representing the information measured by the sensors in an efficient yet self-explanatory way is desirable. As the bandwidth in the wireless sensor network is limited, and the energy available in each sensor node is also limited, the efficiency parameter needs extra attention. The concept of service oriented architecture (SOA) is highly interesting in this context as each measured parameter can be represented as a service to the other nodes, but also globally, as the sensors are connected to the internet using IPv6.

D. SCADA

To replace and extend the HMI functionality provided in the legacy system by an integrated touch panel connected to the PLC, a commercially available SCADA solution was used and configured for the use case. The solution used provides a flexible way of presenting data and configuring the system parameters.

Using an OPC UA client, accessing the server provided by the Mediator, the system can be accessed from anywhere on the connected network rather than the current local access only. At the same time the OPC UA server provides a flexible way to access the system with other standardized tools providing a wide array of possibilities.

IV. IMPLEMENTATION DETAILS

The overall prototype was installed in a similar cabinet as the one installed in the plant in order to facilitate the on-site temporary installation of the prototype.

A. I/O nodes

In the demonstration setup, a total of 14 CoAP services (4 actuators, 6 sensors, and 4 outputs used to indicate system status) were implemented. These were located on three nodes, each executing on a Mulle v6.2 from Eistec [6], equipped with a M16C/62P MCU running at 10 MHz and an 868 MHz low-power IEEE 802.15.4 transceiver.

The software on Mulles was implemented in Contiki, with built-in support for CoAP and 6LoWPAN; support for EXI was added to decrease the size of CoAP packets, which allowed us to avoid fragmentation of CoAP packets and improve robustness of communication. The clock of each node was synchronized to the clock of the edge router using NTP. In order to improve the time synchronization performance, the solution proposed by Keunsol et al. [21] is an interesting approach that needs to be further investigated. The complete communication stack is shown in Figure 7.

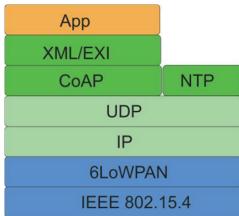


Figure 7: End node communication stack

To enhance the system's security, IPsec is planned to be deployed on the WSAN architecture as well. The use of IPsec on Contiki and 6LoWPAN has been demonstrated by Raza et al. in [22].

B. CoAP and EXI

To implement a SOA-concept in a low-bandwidth 6LoWPAN WSAN used in this application, an efficient compression of the text-based XML service description and data is required. For this purpose EXI (Efficient XML Interchange) [4] was used to represent the XML-based information in binary data format, this reduces the amount of bytes required to represent and transfer the service information. At the application layer, the constrained application protocol (CoAP) is used. CoAP is designed for resource constrained devices like the WSAN nodes used in the demonstration setup.

A key component of the migration of legacy systems to SOA is the use of standard and globally accepted formats for representing the exchanged information. One important result of this demonstration is that it is possible to use EXI for integration of sensor and actuator devices with the SOA automation infrastructure. This enables the implementation of RESTful web services based on CoAP and EXI for industrial application with low real-time requirements.

C. Service Bus

The Service Bus has been implemented on two Raspberry Pi devices running Linux operating system and featuring 512 MB of RAM and 700 MHz ARM CPUs. As illustrated in Figure 6 the main software components of the Service Bus are a pivot data format, a set of connectors acting as external interfaces (DPWS, REST, CoAP, and Modbus), an eventing module, a time synchronization (PTP) module, a logging (syslog) module and the AESOP logic which is reproducing the application logic from the existing PLC.

The two instances of Service Bus dynamically discover each other at startup with WS-Discovery and rely on DPWS for message exchanges between them. A basic cyber-security protection is provided by the combination of Role Base Access Control (RBAC) and user authentication mechanisms.

V. MIGRATION ASPECTS

As proposed in [15] a migration of a large DCS into SOA can be initiated with a smaller step where some key functionality is migrated and the basis of a SOA infrastructure is established in a part of the plant.

This use case provides an example of migration of a number of functional aspects that have been identified in the existing system and provides a minimum requirement of functionality for the SOA-enabled system. Most significant of these are:

1. Local control loop

In the existing system local control is performed within the PLC using internal timers and the pressure switches distributed throughout the system to trigger the start and stop of the lubrication pump and activation of solenoid valves.

In the IMC-AESOP use case this functionality has been distributed primarily to the Service Bus, accessing both the CoAP services provided by the Mulle nodes for sensing and actuating and the AS-i sensors data. The main advantage of the SOA design is to provide added monitoring capabilities on the control loop (timers and sensors data are available as services).

2. Inter-protocol communication

In the existing system there are only two communication protocols involved: The communication to the DCS is handled through Profibus and the collection of data from a large number of field devices is handled through AS-i.

In the demonstrator several new protocols are introduced as part of the architecture to allow communication within the SOA system, while the existing communication interfaces remain accessible through commercially available AS-i and Profibus master modules, respectively. The conversion between different protocols is handled by the Service Bus and the Mediator, as previously described.

3. Alarms and warning

In the existing system alarms are handled through lists of Fault- and Reset-bits with a corresponding list of alarm texts, both in the PLC.

In the SOA solution, those alarms are implemented as events collected from the alarm sources and brokered by the Service Bus. Any interested party can then subscribe to those alarms from the Service Bus. In the demonstrator, the SCADA, the DCS (both through the Mediator) and the Service Bus web client are subscribers of process level alarms. Polling based alarms remain available, which is particularly interesting in a migration context.

4. Operator manual override and Operator configuration

Operator manual override and Operator configuration are the two key functionalities provided by the touch panel HMI in the existing system.

In the SOA alternative, the Service Bus is exposing those two functionalities as services that can be called by any (authenticated and authorized) client application. In the demonstrator two client applications are consuming those services: the SCADA (through the Mediator) and the Service Bus web client.

As mentioned before, the loose coupling provided by this approach can be leveraged in future maintenance operations by allowing replacing transparently and independently either the server or the client part of those services.

VI. VALIDATION RESULTS

A. Functional assessment

The functional validation of the overall architecture was performed on-site during a scheduled maintenance break of the plant. The IMC-AESOP prototype was connected to the lubrication system, by disconnecting the operating cabinet and connecting instead the SOA cabinet.

The lubrication system was then run for several hours for validating the functional behavior of the prototype and collecting timing data.

B. Performance assessment

In order to measure the overall performance of the prototype, the components of the SOA architecture synchronized their time using the PTP protocol (IEEE 1588). All the components were configured to send their logs to a centralized Syslog server (IETF RFC 5424) for timing analysis.

Table 1 below summarizes the average time it takes for a End of line pressure switch event to propagate from the Mulle device to the Mediator through the Edge Router and the Service Bus.

Event	Node	Time offset (ms)
End of line pressure switch	Mulle (sensor)	0
	Edge Router	11
	Service Bus	13
	Mediator	21

Table 1: time measurements

In this example, the CoAP Edge Router receives the event 11 ms after the Mulle detected the end of line pressure

switch, then the Service Bus acknowledges the event 2 ms later and finally the Mediator 8 ms later. The total transmission time between the sensor (Mulle) to the Mediator is 21 ms which is above the current PLC cycle time but stays compatible with the application requirements.

C. Wireless assessment

One parameter of interest that is important for successful deployment of 6LoWPAN devices is the size of the messages that the devices must exchange. Using XML is beneficial for integration of the devices with the data models and message formats used in the upper layers of the automation. By using EXI in strict XML schema mode for the low-bandwidth wireless links, the size of the XML messages is reduced more than 20 times. With that, the size of an EXI encoded digital IO process value with timestamp and quality indicator is 10 bytes as compared to 228 bytes for its plain XML counterpart. Another key performance indicator for wireless applications, especially in noise industrial environments, is the occurrence of retransmissions of packets. A retransmission wastes link bandwidth uses energy and increase latency. During the tests, retransmissions were at a low level, with a stable wireless network as a result.

D. Data modeling

Enabling interoperability of the service specifications and data models is a key technological challenge that SOA systems are aimed to resolve. The full interoperability requires that the syntax and semantic service descriptions are well defined, unambiguous and enable dynamic discovery and composition. Thus far, most if not all SOA installations are enabling pure syntax interoperability with little or no support for standard based semantic descriptions. The use of structured data formats only partially resolves the problem by supplementing the exchanged data with meta-information in the form of tags and attributes in the case of XML/EXI for example. The tag names are ambiguous and usually insufficient to describe the service functionality in full.

Applying application level data model standards is often used as a solution to that problem as the syntax to semantics mapping is predefined. Example of such standard is Smart Energy Profile 2 that clearly states the physical meaning of the tag names and structures defined for the service messages in the domain of energy management. One problem when complying with such standards is that they are almost always domain specific which requires mapping of the semantic descriptions from one standard to all others in use.

Another approach is to define generic semantic data model that is applicable to wide range of use cases. This is the approach selected for the work presented in this paper. The initial investigation highlighted the Sensor Model Language (SensorML) [18] developed by Open Geospatial Consortium (OGC) as a promising specification for generic semantic description of sensory data. However, the complexity and size of SensorML specification limit its use to more capable devices. Small scale experiments with a number of sample SensorML messages showed that even EXI representation will not be sufficiently small to fit a battery powered wireless sensor nodes that have low-power, low-bandwidth radios.

Another possible specification for sensor data is the Sensor Markup Language (SenML) IETF draft [19]. It has a very simple design that is consistent with RESTful architecture and is targeted at resource-constrained devices. The evaluation of SenML specification showed that it meets the requirements for hardware utilization but there are areas that are too much simplified and insufficient to describe the data in the details required by the target application. Example of such limitation is the precision of the time stamping of the sensor data – SenML allows for up to seconds resolution that is not enough for most use cases. This led to the use of custom generic data representation that is reusing many of the design choices in SenML.

E. Overall drawbacks and benefits

A general drawback of the proposed solution is obviously its lack of maturity, in a sense that it consisted in a set of prototypes provided by different partners, none of them being industrialized yet. This translated into both unreliability issues and integration complexity. Part of the integration difficulties consisted in having a specific configuration and monitoring interface for each partner component.

This heterogeneity and relative complexity of the demonstrator can in turn be perceived as an opportunity to validate the SOA approach, each component of the architecture exposing and consuming services to/from other components, with a fairly high level of loose coupling.

In a productized version of the demonstrator, all middleware components (Mediator, Service Bus, Edge Router and potentially AS-I gateway) would ideally be merged into one product, thus reducing the main complexity of the system. However, a projection to a productized version of the SOA middleware would still lead to a higher level of internal complexity compared with a less versatile PLC based solution.

The main benefit of the proposed solution, compared with the installed solution, is to facilitate the overall system installation and maintenance. Although the installation benefit was not obvious on the demonstrator due to the multiplicity of technologies (and partners) involved, the maintenance and monitoring value was fairly obvious thanks to the advanced monitoring capabilities provided by the added services and displayed through the SCADA (timers, sensors values, alarms...).

VII. CONCLUSION

The on-site validation of the IMC-AESOP prototype provided very positive feedbacks considering that both functional and performance results were in line with customer expectations, combined with added supervision and control capabilities at the SCADA level.

SOA proved to be valuable both at device and application level by providing a high level of loose coupling between the various components of the system. Eventing completed nicely the SOA architecture by reducing the overall latency of the information flow.

On the wireless side, the tests show that CoAP-based services over 6lowPAN can be used for process monitoring and control applications with no low-latency requirements.

More research is needed though in order to improve both scalability and robustness and minimize latency.

ACKNOWLEDGMENT

The authors would like to thank the European Commission for its support through the European Collaborative Research project IMC-AESOP of the FP7 program, which funded this activity.

The authors would also like to thank the LKAB Company for the open collaborative work on this solution and by giving us access to their Kiruna facilities.

REFERENCES

- [1] OASIS, “Devices Profile for Web Services Version 1.1”, 2009, <http://www.oasis-open.org/committees/ws-dd>
- [2] SOA4D Forge - Service-Oriented Architecture for Devices, <https://forge.soa4d.org/>
- [3] Carl Hewitt, Peter Bishop and Richard Steiger: A Universal Modular Actor Formalism for Artificial Intelligence. International Joint Conference on Artificial Intelligence, 1973
- [4] Efficient XML Interchange (EXI) Format 1.0 W3C, 2011
- [5] Kuladinitthi, K.; Bergmann, O.; Pötsch, T.; Becker, M. & Görg, C. Implementation of CoAP and its Application in Transport Logistics Proc. IP+ SN, Chicago, IL, USA, 2011
- [6] Hartke, K. Observing Resources in CoAP CoRE Working Group, 2013
- [7] Kovatsch, M.; Duquennoy, S. & Dunkels, A. A low-power CoAP for Contiki Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on, 2011, 855-860
- [8] Kyusakov, R.; Eliasson, J. & Delsing, J. Efficient structured data processing for web service enabled shop floor devices Industrial Electronics (ISIE), 2011 IEEE International Symposium on, 2011, 1716 -1721
- [9] Eliasson et al. The Titanium IoT service composition framework. April 2013 <http://sourceforge.net/projects/titaniumiot/>
- [10] Castellani, A.P.; Bui, N.; Casari, P.; Rossi, M.; Shelby, Z.; Zorzi, M. Architecture and protocols for the Internet of Things: A case study. 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010
- [11] Karouskos, S. ; Colombo, A.W.; Jammes, F.; Delsing, J.; Bangemann, T., Towards an architecture for service-oriented process monitoring and control, in IECON 2010.
- [12] The Mulle sensor and actuator platform. <http://www.eistec.se>
- [13] <http://www.lkab.com/en/About-us/Overview/Operations-Areas/Kiruna/>
- [14] <http://www.as-interface.net/>
- [15] Delsing, J. et al, Migration of industrial process control systems into service oriented architecture, IECON 2012.
- [16] Shelby, Z.; Embedded web services, Wireless Communications, IEEE, 2010, 17, 52 -57
- [17] Feldhorst, S.; Libert, S.; ten Hompel, M. & Krumm, H.; Integration of a Legacy Automation System into a SOA for Devices in Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on, 2009
- [18] Sensor Model Language (SensorML) Implementation Specification, Standard, OGC, 2010
- [19] Jennings, C.; Shelby, Z. & Arkko, J. Media Types for Sensor Markup Language (SenML), Standard, 2013
- [20] Bormann, C.; Castellani, A.P.; Shelby, Z.; CoAP: An Application Protocol for Billions of Tiny Internet Nodes, in IEEE Internet Computing, vol.16, no.2, pp.62-67, March-April 2012.
- [21] Keunsol Kim; Seung-Woo Lee; Dae-geun Park; Bhum-Cheol Lee, PTP interworking 802.15.4 using 6LoWPAN, in Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on , vol.01, no. , pp.873,876, 15-18 Feb. 2009
- [22] Raza, S.; Duquennoy, S.; Höglund, J.; Roedig, U and Voigt, T.; Secure Communication for the Internet of Things - A Comparison of Link-Layer Security and IPsec for 6LoWPAN, in Security and Communication Networks, Wiley, 2012

PAPER C

Migration of Industrial Process Control Systems into Service-Oriented Architectures

Authors:

Oscar Carlsson, Jerker Delsing, Fredrik Arrigucci, Armando W. Colombo, Thomas Bangemann and Philippe Nappey

Reformatted version of paper submitted to:

Journal paper, International Journal of Computer Integrated Manufacturing, 2016.

© 2016 Taylor & Francis. Reprinted, with permissions, from Jerker Delsing, Fredrik Arrigucci, Armando W. Colombo, Thomas Bangemann and Philippe Nappey, *Migration of Industrial Process Control Systems into Service-Oriented Architectures*, Taylor & Francis International Journal of Computer Integrated Manufacturing, 2016.

Migration of Industrial Process Control Systems to Service Oriented Architectures

Oscar Carlsson, Jerker Delsing, Fredrik Arrigucci, Armando W. Colombo, Thomas Bangemann and Philippe Nappey

Abstract

The use of Service Oriented Architectures (SOAs) in industrial automation promises an improved cross-layer integration as well as a functionality decoupled from the technical implementation. Compared with the earlier investigated manufacturing industry, control systems in the process industry reveal additional challenges in terms of migration from a legacy control system to a SOA control system.

The successful migration of a highly integrated process control system, without reducing reliability or availability and, at the same time, preserving functionality and productivity, requires a detailed plan and certain specialized technology.

This paper presents the challenges in the migration of a process control system and proposes a structured method for migration. The migration procedure proposed comprises four steps: initiation, configuration, data processing and control execution. A technology demonstration at a pelletizing plant illustrates how the first of these steps could be implemented.

1 Introduction

A demand exists in the process industry for the modernization of process control equipment as ageing process control systems are becoming increasingly maintenance intensive [1]. In addition, older systems lack certain technical capabilities and features offered by newer equipment. Service Oriented Architecture (SOA) is regarded as a promising candidate for supporting cross-layer integration that will render distributed systems more interoperable. A shift to Service Oriented Architectures has been occurring at the enterprise system level for many years. This technology was pioneered by IBM and others, and important contributions have been made by many sources; examples can be found in [2]. Some early examples of the application of SOA to industrial systems are presented by [3], and several works have been published on the use of SOA in Systems of Systems engineering [4, 5].

Several European collaborative projects, e.g., SIRENA [7], SODA [8], SOCRADES [9, 10, 11, 12], and others [13], have demonstrated the feasibility of embedding Web Services at the device level and integrating these devices with the enterprise systems typically used in industrial automation [12, 14]. Other projects that indicate the same development of industrial control systems include the European Initiatives PLANTCockpit [15, 16],

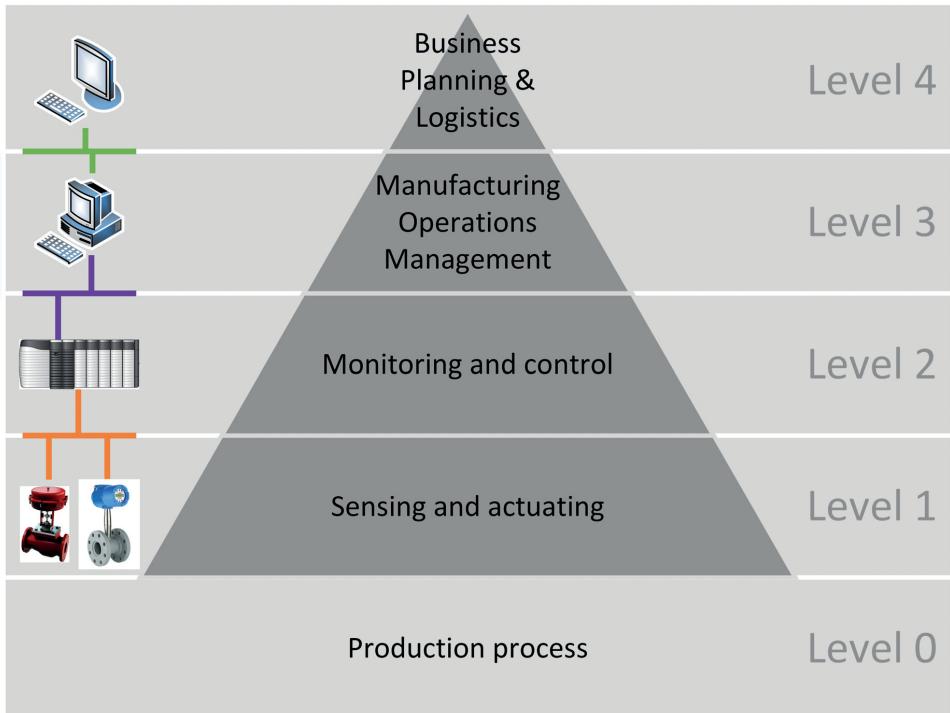


Figure 1: ISA95 architecture of an automation system and functional hierarchy according to [6]

17], KAP [18, 19], ActionPlanT [20], and the European Embedded Systems Platform Advanced Research & Technology for EMbedded Intelligence and Systems - ARTEMIS [21].

These projects primarily addressed factory automation, whereas the IMC-AESOP project [22, 23, 24] considered the application of the SOA paradigm to address the next generation of Distributed Control System (DCS) and Supervisory Control And Data Acquisition (SCADA) systems, with a major focus on the process industry and industrial process control systems. The SOA paradigm applied to control and automation yields technologies, methods and tools that can enhance interoperability by decoupling the functionality from the technical implementation. As a result, the transparency of the entire infrastructure is increased.

This paper builds on the results of the IMC-AESOP project and details a migration process where the target architecture is the one presented as the IMC-AESOP Approach [25]. This approach is anticipated to enable possibilities for new functions and easier integration of systems [26], as well as the benefits of flexibility and reconfigurability more generally associated with Service Oriented Architectures [27]. Within this scope, the term “legacy systems” refers to traditional industrial process control systems that are

not based on the principles of SOA. These typically include proprietary protocols and interfaces tied to the hardware and communication infrastructure, resulting in vendor lock-ins and possibly site-specific solutions.

Legacy systems in the domain of industrial process control are typically implemented following the five-level model defined within the ISA 95 / IEC 62264 standard (illustrated in Figure 1). Operations defined by this standard are inherent to established production management systems [6]. In this context, concepts for integrating legacy systems (specifically at lower levels) into SOA-based systems can be seen as business enablers that will move the customer from where she/he is today [24] into the future.

The migration process presented here is based on maintaining functionality throughout the migration and in the fully migrated system. This perspective describes the functionality identified in existing systems and how it can be preserved throughout the four steps of the migration procedure: initiation, configuration, data processing and control execution. The authors argue that these steps are necessary for the successful migration of the DCS and SCADA functionality into a service-based automation cloud. To enable the integration between legacy and SOA components, some critical migration technology is proposed based on the Mediator and other concepts.

The novelty of migrating from a legacy process control system to a Service Oriented Architecture lies in the gradual upgrade of highly integrated and vendor-locked standards to a more open structure while maintaining functionality. The challenges in the step-wise migration of a highly integrated vendor-locked DCS or SCADA are discussed.

2 Migration

The migration of industrial process control systems to a new platform or technology will be a large undertaking, considering the size and complexity of most existing systems. A migration may be initiated for different reasons in different cases but, as with any large project, it is important to have a well-defined purpose and justification. The purpose and justification may vary as, e.g., some users may want certain functionality such as e-maintenance, others may want to have systems compliant to certain standards, whereas yet other end users might obtain SOA components virtually without knowledge of it, e.g., if SOA components are integrated with other delivered functionalities.

Afterwards, it is important to evaluate the migration and assess whether the requirements were fulfilled. Therefore, the requirements must be quantifiable and measurable. For example, to minimize the negative impact of migration, measurable requirements must exist for effects such as downtime, control problems, costs, interoperability and performance as well as more qualitative aspects, i.e., personnel training and employee satisfaction.

The proposed migration procedure is based on the work by [28] and further details how an industrial process control system can be migrated to SOA, taking into account the functional aspects of a large industrial system. Although the proposed procedure goes into greater detail than any published work the authors are aware of, it is not as detailed as some existing strategies for migrating enterprise systems [29, 30] where SOA

has been established for some time.

As a basis for the migration procedure, a set of expectations for migration to SOA was identified by [28], summarized here as follows:

- The new architecture and migration strategy must ensure the same level of reliability and availability as the legacy system.
- The migration procedure must not induce any increased risk of reduced reliability or availability for the equipment, affected staff or production process.
- After the migration, the plant must still provide the same or better processes, extended service life of the plant process equipment, adequate information and alarms depending on the department and personnel skill and improved vertical (cross-layer) communication with additional information available at the plant-wide level.
- The migration must provide the same or improved functionality in all aspects of plant operation throughout the complete migration.

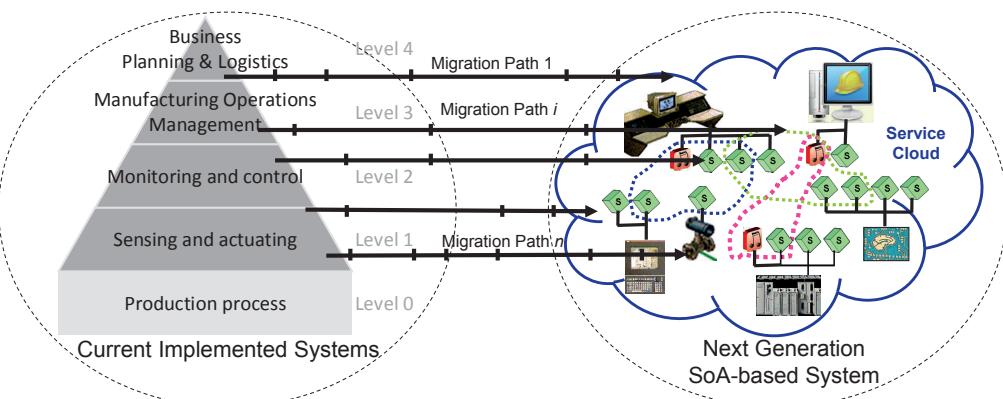


Figure 2: Migration approach from legacy to SOA-based systems.

This migration approach from the current legacy to a SOA-based industrial system is expected to follow a different migration paths for different sections – each with its set of steps, as visualized in Figure 2.

2.1 ISA 95 - IEC 62264

As used in process or manufacturing automation, today's control systems are typically structured in a hierarchical manner, illustrated in Figure 1.

The IEC 62264 (based on ISA 95) is an international standard for the integration of enterprise and control systems [6] and was developed to provide a model that end users, integrators and vendors can use when integrating new applications into the enterprise. The model aids in defining boundaries between the different levels of a typical industrial enterprise.

The ISA 95/IEC 62264 defines five levels, with **level 0** as the industrial process to be controlled. **Level 1** connects the control system to the process with sensors and actuators. Through the sensors, the control system can receive information on the process and subsequently regulate the process through the actuators. At **Level 2**, distributed monitoring and control enable plant supervisory control. In the field of automation in the process industry, there are two main system design strategies, the Distributed Control System (DCS) and Supervisory Control And Data Acquisition (SCADA). The DCS is highly integrated compared to a SCADA solution, which is standard in factory automation. The SCADA is a supervisory system that communicates through open standard protocols with subordinated Programmable Logic Controllers (PLC) or Remote Terminal Units (RTU). The PLCs in the SCADA solution are autonomous compared to their counterparts in the DCS, which are often referred to as controllers.

The operational management of the production is performed at **Level 3**, in which Manufacturing Execution Systems (MES) provide multiple information and production management capabilities. In the context of control hierarchy, the main function is plant-wide production planning and scheduling. Level 3 integrates information from production and plant economics and provides a detailed overview of the plant performance. At **Level 4**, Enterprise Resource Planning Systems (ERPs) are typically installed for strategic planning of the overall plant operation according to business targets.

Several providers of current Level 4 systems (e.g., enterprise systems, as shown in Figure 1) already support service-driven interaction via Web Services. The challenge of vertical integration of enterprise systems was successfully addressed by Kalogerias et al. [31]. These systems may support event-driven interaction, and Service Oriented Architecture is an approach applied at this level. Services are also used for integration between Level 3 and Level 4 systems that are available on the market. OPC UA [32] is a spreading-up technology used for this purpose, and in close cooperation with the OPC Foundation, PLCopen defined an OPC UA Information Model for IEC 61131-3. A mapping of the IEC 61131-3 software model to the OPC UA information models was defined and led to a standard method for how OPC UA server-based controllers expose data structures and function blocks to OPC UA clients such as Human-Machine Interfaces (HMIs) [7]. The OPC UA relies on Web-Service-based communication. In 2011, a working group was established to focus on the definition of communication mechanisms via OPC UA for MES integration of Level 2 systems as well as the definition of semantics for MES integration. These activities can be viewed as attempts to move towards the use of common technologies across different levels of production systems. If successful

and widely adopted, the usage of Web-Service-based communication between systems at Level 2 and Level 3 should further increase the want for a migration of the process control systems at lower levels.

2.2 Challenges in migrating industrial process control systems

The ISA 95/IEC 62264 defines four levels of systems, and for each of these four levels, certain problems and challenges become imminent upon considering their implementation using a SOA-based approach.

Migration into SOA at Levels 3 and 4 does not differ significantly for factory automation and process control systems [10].

At Level 2, certain non-resolved challenges of migration exist in the process industry. In this paper, the process control system is defined as a DCS including HMI workstations, controllers, engineering stations and servers, all linked by a network infrastructure. A DCS is truly ‘distributed’, with various tasks carried out in widely dispersed devices. Migration of Level 2 functionality in the form of a DCS offers challenges for co-habitation between legacy and SOA systems as well as the migration of the control execution [23, 33]. In this paper, the DCS is exemplified by a server/client-based system, as depicted in Figure 3, which is a common topology.

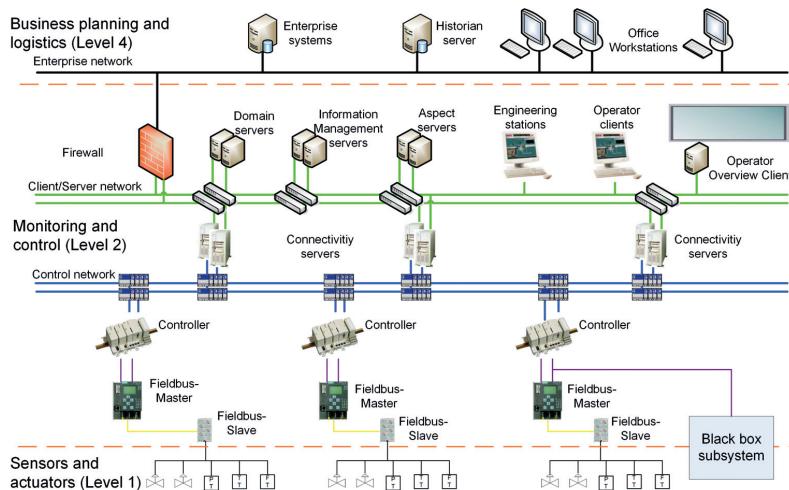


Figure 3: Legacy system architecture

To migrate a highly integrated DCS, the following challenges should be addressed:

- **Preservation of functional integration:** There are advantages to a highly integrated DCS, which provides a tight link between the HMI and control execution.

Thus, design engineering, commissioning and operation can be pursued in a significantly more uniform manner. For instance, the HMI and control execution can be configured using the same tools, which facilitates conformity. These advantages must be maintained, although the integration is broken down and substituted by open standards.

- **Grouping of devices:** Within a given system, which devices should be migrated to SOA as devices and which devices should be grouped together and migrated to a SOA must be determined. For example, a subsystem that uses feedback and regulation might require legacy interfaces because of real-time demands, and therefore, such a group of devices should be given a SOA interface for the group using a Mediator and should not be treated at the device level. This portion of the system may be treated as a ‘black-box’.
- **Preservation of real-time control:** The real-time control execution, which is secured in the controllers in the legacy system, must be preserved.

The migration of Level 1 devices proposes the use of one of two approaches depending on the nature of the devices and their functionality:

- Individual devices can be exchanged or retro-fitted to provide a system of individually SOA-compliant devices.
- Devices can be wrapped as a complete legacy subsystem with a common SOA interface to provide the functionality of the subsystem as a service to the SOA-enabled service consumers.

It is envisioned that both of these

Migration of an industrial system structured according to ISA95 to SOA generally cannot be performed at only one level of the architecture shown in Figure 1 because specifications and system characteristics at a defined level are closely related to specifications at other levels. Thus, the migration strategy must address how the migrated component can represent the legacy functionality and how it is involved with other levels of the control system.

3 Functional aspects

To present a broader picture of the migration process, it is important to address the technological difference in preserved functionality between legacy and IMC-AESOP systems, not only in terms of individual architectural components but also in terms of the functionality that can be achieved through the combination of components. This chapter will focus on describing the functionality provided by legacy systems, identifying the functionalities that are critical and explaining how those functionalities are maintained or improved during (and after) migration to IMC-AESOP architecture.

- **Local control loop:** The function of a local control loop refers to the low-level automated control that regulates a certain portion of the plant process with a relatively low number of actuators and sensors. The control may be continuous or discreet and may use analogue as well as digital actuators and sensors. In many cases, the control will require low latency and short sample times, resulting in high bandwidth.

An example is to control the flow of a liquid through a pipe, using sensors to measure the flow and actuators (e.g., a pump) to increase or decrease the flow based on the measured values.

- **Distributed control:** This function refers to all forms of control in which parts of the control loop are located far away from each other geographically or architecturally, meaning that the control cannot be executed by a single device (controller) with direct access to both sensors and actuators. An example is to increase the material flow into a large system because measurements inside the system indicate a need for more material.

- **Supervisory control:** This form of control is often executed at a higher level based on information from more than one subsystem and is usually much slower than the local control loop. Often, the supervisory control has no direct access to the sensors or actuators but uses aggregated process values as input and actuates by changing the set point of a local control loop.

An example is to decrease the throughput of a production process due to the warehouse being full.

- **System aggregation:** Low-level devices and subsystems are often presented to higher level systems in an aggregated form to provide an understandable overview of the system to operators, engineers and others working with the system.

An example is that a set of pumps and sensors that control the flow through a pipe are aggregated into a pump group and display performance, events and alarms as one object at higher levels rather than individual sensors, pumps, and drives, among others.

- **Inter-protocol communication:** Because different levels of the DCS use different communication standards and protocols, all communication between components that are not on the same or neighbouring levels must pass the information through one or more other components. These other components must therefore be able to interpret or translate the information between the different standards and protocols. The effort required for this type of communication varies greatly depending on the standards and protocols involved.

An example is that a PLC may have to convert information received through a fieldbus network from a subordinate system to an alarm that can be sent to a supervisory system across a control network using a different protocol.

- **Data acquisition, display and storage:** Process and system data gathered at all levels of the DCS must ultimately be made available to operators and other connected systems. The availability of correct data is vital to both operators and management to optimize performance and analyse anomalies. In certain cases, historical data storage is integrated into the DCS, but even in these cases, the functionality is not an integral component of the DCS functionality and can be treated as a peripheral system.

An example involves data from a pump group, which must first be acquired from sensors, actuators and controllers and then aggregated, displayed to operators in a control room and stored in a secure location for historical access.

- **Alarms and warnings:** All systems have methods of indicating process anomalies to the personnel working with the process. In a well-developed DCS, many functions related to alarms and warnings allow the distribution of the information to the appropriate staff and contain several modes of suppression and acknowledgement of alarms and warnings.

An example is that some alarms from an object should be displayed immediately to an operator, while others are shown only if certain conditions are met; warnings may be sent primarily to maintenance personnel and only displayed to operators if actively requested.

- **Emergency stop:** The emergency stop is a vital component of most process control systems and is often regulated by national laws and regulations. In a large process plant, the emergency stop may be much more complex than simply shutting off the power to all components because this action may cause situations in which a build-up of heat or pressure or a chemical reaction would cause a greater disaster than if the plant were kept running. It is important that a process control system is able to execute a reliable shut-down procedure even in unexpected situations.

An example would be to stop a pump group because an emergency stop was pressed in that room, but if the pump group was supplying a required coolant, it is also necessary to automatically start a redundant pump group so that the process does not overheat.

- **Operator manual override:** In most plants, an operator is required to be able to control parts of the system manually via an HMI to handle irregular or unexpected situations. This option may be intended to support maintenance operations in which systems are disconnected in a controlled manner or for use if the operator must address unexpected faults in the process or in the automation system.

An example is that during a maintenance operation in which the system should be kept running but some sensors are disconnected, it may be necessary for an operator to assume control of the system using visual input or extrapolating from other measurements to ensure acceptable performance.

- **Operator configuration:** Most operator stations allow alteration of selected parameters in the system, i.e., the plant or system operation mode or control set points for subsystems, based on information not available in the automation system.

An example is that if operating conditions, such as market situations or forecasts, change in a way that is not handled automatically by a supervisory system, it may be necessary for an operator to manage the process throughput by changing some parameters in the control system.

- **User management and security:** Because many components of the DCS are interconnected and many people with different roles work with a DCS, it is important that each person is presented with a level of information that is sufficient and relevant to their role. To limit human errors as well as malicious actions, it is important that all personnel are authenticated for the role in which they are allowed to access the system. The authentication may not always be limited to software but may instead consist of limiting physical access to certain areas or stations.

An example is that all users have individual usernames and passwords for accessing the system, which then limits the access to critical information. In addition, certain hardware, such as an engineering station, may be kept in a separate location accessible only to authorized personnel.

4 Migration procedure

The migration from today's state-of-the-art control systems to those that exploit upcoming technologies has become a crucial area of innovation. Migration and integration are complementary processes, and a migration process may contain several steps of integration while potentially exploiting different integration technologies.

Considering the layout of a server/client-based SCADA/DCS, a stepwise migration through four major steps is proposed. The four major steps can contain sub-steps and may be spread out over a long period of time, but each major step should be completed before the following step is initiated. The four major steps suggested are the following:

- Initiation
- Configuration
- Data processing
- Control execution

During the entire migration, the system will require one or more Mediators to allow communication between the SOA components and the portions of the legacy system that have not yet been migrated. The propagation of the Mediator and the growth of the SOA infrastructure, represented by the cloud in the figures, are applied to the legacy SCADA/DCS presented in Figure 3. Putting an emphasis on the DCS portion, Figures 4

through 7 present the different resulting stages obtained throughout the entire migration process.

Mediators and Gateways

The use of Gateways or Mediators is a proven concept for integrating/connecting and migrating devices attached to different networks. The concept can be used to transform protocols as well as data syntax. Although full semantic integration is difficult to achieve, it is possible to use a Mediator to facilitate the transformation between data-centric approaches, such as those typically followed by field-bus concepts, and service-oriented, event-centric approaches.

The Mediator concept [34, 35] discussed in this work is based on the Gateway concept with additional added functionality. Originally intended to aggregate various data sources (e.g., databases, log files, etc.), the Mediator components evolved with the advent of Enterprise Service Buses (ESBs) [36]. Currently, a Mediator is used to aggregate various non WS-enabled devices or even services in SOAs. Using Mediators instead of Gateways provides advantages in management of semantics or performing pre-processing of data from legacy networks, e.g., representing a specific subsystem. Due to the diversity of data (e.g., the quality, quantity and granularity of data) requested by different applications, interface devices will normally be built as a combination of Gateway and Mediator. Because it may also be applicable in integrating service-oriented sections (e.g., the retrofit of a plant section or replacement of a specific subsystem) into existing systems, this Gateway and Mediator concept can be extended to represent services in data-centric systems (today's legacy systems). The Mediator and Gateway concepts are both powerful means for integrating single legacy devices or legacy systems by encapsulating 'isolated' functionalities.

Specific configuration efforts for the Mediator or Gateway itself will be necessary during the migration, as the requirements change when subsystems are migrated to SOA.

4.1 Step 1: Initiation

The initial SOA 'cloud' requires a subset of the basic services presented by [37] to support basic communication and management of the cloud. At this point, **User management and security** must be considered, although depending on the scenario, it may not be necessary to implement a full company-wide security framework if it is implemented in a manner that can later be extended to the entire company.

Once the fundamental parts of the service oriented architecture are in place, the first peripheral subsystems can be migrated, and new components can be integrated into the SOA. For the migration of subsystems as well as the integration of new components, certain features must be implemented to address the limitations of the Mediator and its communication paths. Precisely what features that are necessary will depend both on the implementation of the mediator and the requirements migrated systems. Some features that are likely to be required are listed in subsection 4.1, and described in more details in section 5.

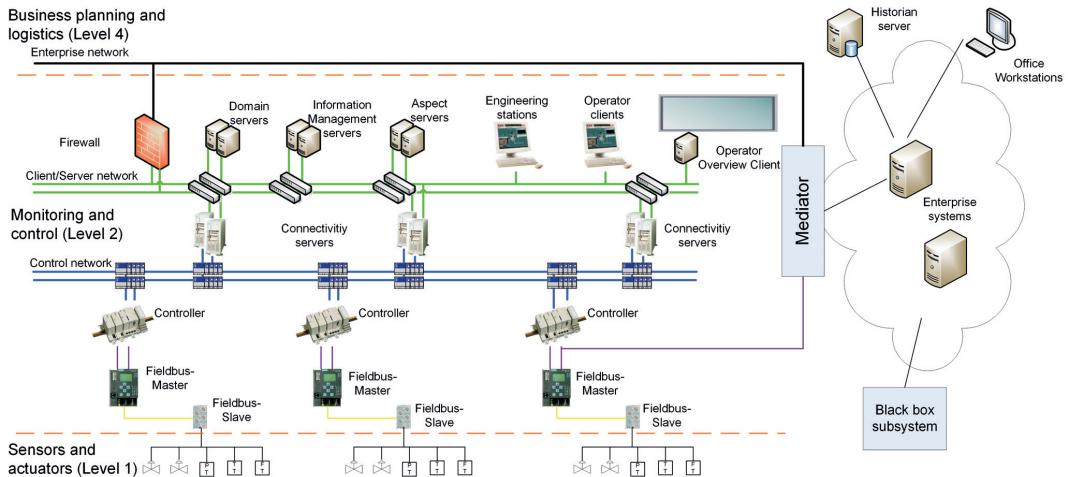


Figure 4: DCS after the first step of migration

The systems migrated in this step include the following sub-systems that are not direct components of the highly integrated DCS:

- Low-level black boxes
- High-level systems for business planning and logistics, such as maintenance systems

Migration is limited to the operational phase of the integrated systems. The engineering required for this step lies beyond the scope of migration. An appropriate engineering approach dedicated to this migration step implies multi-step configuration as follows:

- Configuration of every legacy system including the legacy interface within the Mediator
- Configuration of the SOA system
- Configuration of model mapping within the Mediator

Exploiting machine-readable legacy configuration information would be helpful at every step. Today, configuration information is available through different technologies, e.g., GSD, DD, and paper documentation. This type of information is mostly available for single devices. Engineering stations take this information as input and generate system configuration information in proprietary formats.

Status after the first step

At this point, several components of the functional aspects can be considered to be at least partially migrated. Most likely, a subset of the **Local control loop** functionality is migrated. **Inter-protocol communication** is required both in the migrated and the traditional portions of the system, and **User management and security** must be at least partially implemented in the SOA-system without compromising existing security or creating unnecessary obstacles for users or user administrators. **System aggregation, Emergency stop, Alarms and warnings, Operator manual override** and **Operator configuration** have all been implemented in the SOA system to the extent required by the migrated subsystems, whereas the respective functionality in the traditional system is virtually untouched.

4.2 Step 2: Configuration

This is the first step where components that are heavily integrated into the DCS are migrated. The purpose of this step is to migrate portions of the DCS that do not require short response times or the regular transport of large amounts of data (please refer to Fig. 5). The majority of functions that qualify for this migration step are tangentially concerned with the configuration of different portions of the DCS. The point of origin for most (if not all) configurations is the Engineering Station (ES), which is used for the engineering and configuration of most portions of the DCS.

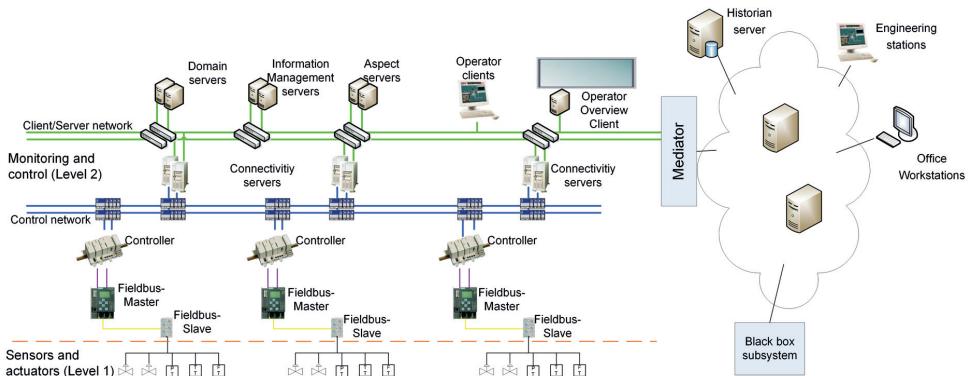


Figure 5: DCS after the second step of migration

As the ES is migrated to SOA, a major increase is observed in the number of services that the Mediator must supply to the SOA cloud because it must represent configuration aspects of all legacy systems and devices not yet migrated and allow the configuration of all systems and devices in addition to the operational data migrated in the first step. This means that the configuration of low-level devices and control is carried out on the

ES in the SOA environment using configuration services provided by the Mediator; the configuration is subsequently compiled by the Mediator into the respective legacy formats and downloaded into the legacy controllers. The configuration of HMI, faceplates and associated systems is similarly carried out in SOA and converted by the Mediator to a format that can be downloaded into the legacy aspect servers and other legacy systems. The configuration of legacy devices from SOA might also require the Mediator to be able to extract legacy designs and configurations that may be stored in the aspect servers or controllers so that old designs can be reused and modified by the SOA engineering stations.

Because legacy systems usually do not provide sufficient meta-data, sufficient configuration information cannot necessarily be extracted by a Mediator from the installation (legacy systems). Today, configuration information is available through different technologies (GSD, DD, paper documentation, etc.), and this type of information is mostly available for single devices. Engineering stations take this information as input and generate system configuration information in proprietary formats that are not necessarily interpretable by other tools.

Consequently, for the overall engineering, a SOA engineering station should be able to import the relevant configuration information of different legacy systems in addition to the limited capabilities provided by the Mediator itself. If such a tool were available, one could design a Mediator that acts as a configuration station for different legacy systems (compiles configuration information into legacy formats) while receiving basic configuration information from the SOA engineering station.

Status after the second step

Because most of the functionality of everyday operations should be unaffected by the migration of engineering and configuration tools, only a small portion of the functionality discussed in the section Functional aspects is affected. Most notably, there will be an increased need for **Inter-protocol communication**, and there is the possibility of using more of the functionality categorized as **Supervisory control**. In addition to the migration of the engineering station, this step means that certain additional parts of **User management and security** are migrated, but apart from these, most functional aspects should be similar to those of the first step in the migration.

This approach may be combined with the multi-step configuration described in the previous step.

4.3 Step 3: Data processing

In this third step, the migration includes all components and/or subsystems that do not require the short response times (millisecond range) currently unachievable by the SOA technology (please refer to Figure 6). This category of components includes Operator Clients (OPs) and Operator Overview Clients (EOWs) as well as Aspect Servers (ASs) and Information Management Servers (IMs). Because all points of user interaction with the system are now moved to the SOA, this means that the legacy Domain Servers (DSs)

are redundant. However, because user management and security must be available in SOA from the first step of the migration, there is likely no need for the Domain Servers in the SOA cloud, although this functionality can be considered as migrated.

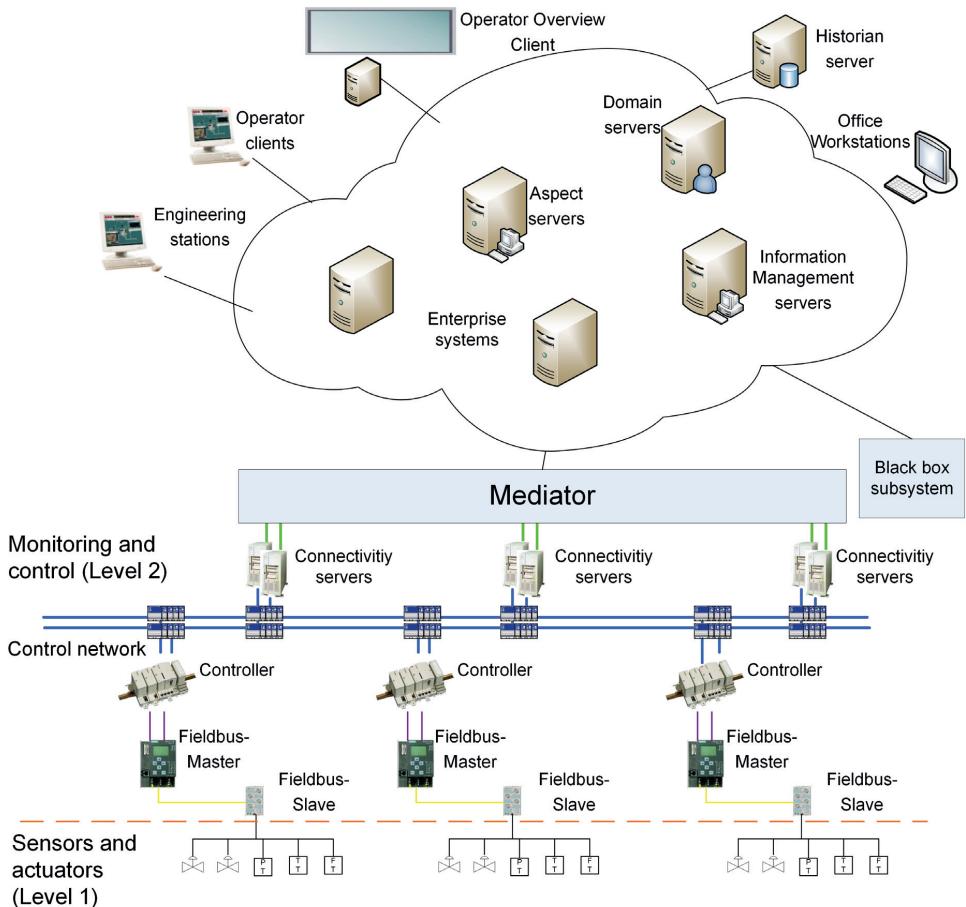


Figure 6: DCS after the third step of migration

The migration of the Operator Clients and the Aspect and Information Management servers implies that the role of the Mediator is once again fundamentally changed. In Step 3 of the migration, there is less need for a flexible Mediator that can communicate with many different legacy components, and the new requirements more concern the need to present large amounts of data available from legacy controllers to the migrated Operator clients and other data processors and consumers. This activity is closely related to the

purpose of the Connectivity Servers (CSs), and it is suggested that the Mediator in Step 3 is implemented as a new interface in the Connectivity Servers.

Status after the third step

At this stage, several operator-centric portions of the functionality are completely migrated. Most significantly, **Operator manual override** and **Operator configuration** are fully migrated. All of **Data acquisition, display and storage**, except for the first level of the acquisition of data from the devices up to the controllers, are also migrated at this step. Because the functionality for data acquisition is migrated, selected additional functionalities for **System aggregation** might be required to present the data from underlying systems for the case in which this need is not sufficiently covered by the traditional systems. In addition, all of the **Alarms and warnings** functionalities, apart from the selected generation of alarms at the controller level, are migrated together with **User management and security**.

4.4 Step 4: Control execution

In the fourth and final step of migration, the functionalities traditionally provided by controllers must be migrated (please refer to Figure 7). Because control execution in the legacy system can be grouped together with several control functions in one controller, or in certain cases spread out with different portions of a control function executed by more than one controller, it is of utmost importance that control execution is migrated function-by-function rather than controller-by-controller.

Depending on the performance requirements of each control function, there may be a need for different strategies for different functions. In the cases in which SOA-compliant hardware is available for all functions, an active migration may be suitable, in which a detailed schedule can be designed over the migration of all functions to enable a controlled migration towards a set deadline. In other cases, it may be more suitable to allow legacy controllers to fade out as functions are migrated in the course of the normal maintenance and lifecycle management of the plant. The fade-out option means that Step 4 of the migration may require a long duration, but this process may save costs if legacy devices are used for their full lifetime, whereas most benefits of the SOA are already available.

Status at the end of the fourth step

During this fourth step, most of the migrated functionality relates to control at a certain level because most of the monitoring, engineering and administration has already been moved to the SOA system. In particular, this step relates to the migration of the **Local control loop**, **Distributed control** and **Supervisory control**. Another key function that is migrated in this step is the **Emergency stop**, which can be considered as a form of human-in-the-loop control with very specific conditions. As each specific control function is migrated, so are the related support functions, i.e., **System aggregation**, **Data acquisition, display and storage** and **Alarms and warnings**.

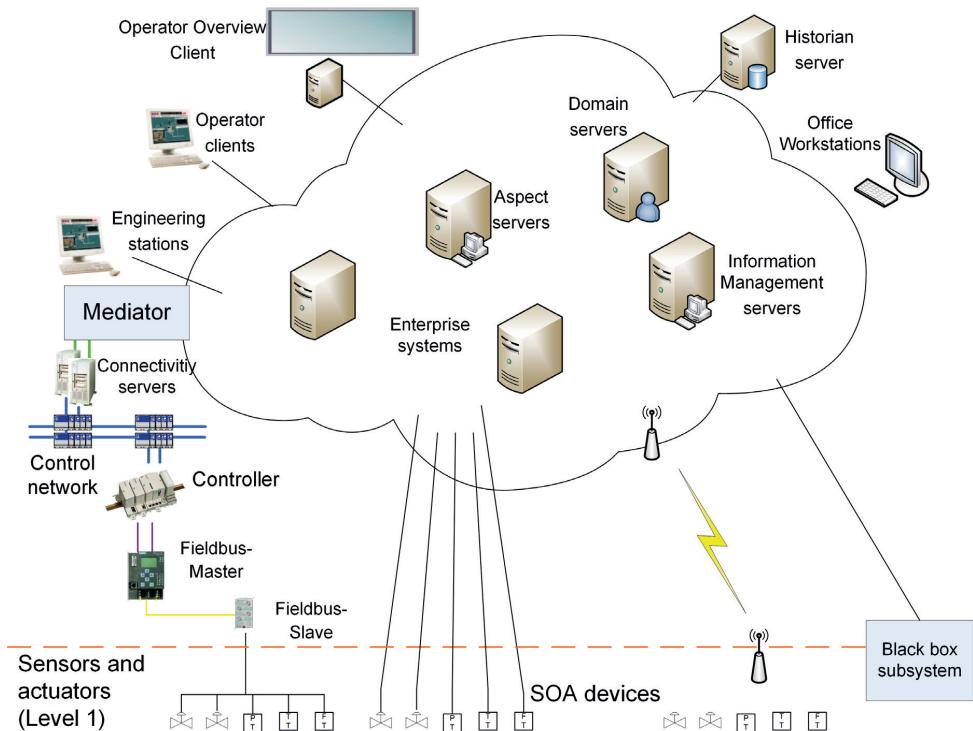


Figure 7: DCS after the fourth step of migration

5 Migration of functionality

To ensure and support the preservation of functionality throughout the migration process, each functional aspect identified in a DCS has been analysed, and an example is presented for each aspect to explain how the migrated system could provide the functionality in question. These examples are not the only possible implementations of the functionality, but they should provide examples that cover the complete DCS, thus illustrating how functionality can be preserved or improved throughout the migration. Whether these examples are the most suitable implementations will depend on the application and system requirements, but they have been deemed useful illustrations of the implementations.

5.1 Local control loop

In the case where a local control loop is performed within a loosely connected subsystem, it can be useful to migrate that specific instance of this functionality in the first migration step itself (**Initiation**). In most scenarios, however, local control loops are executed in a heavily integrated low-level controller and are better left to the final step (**Control**)

Execution) of the migration.

Legacy implementations use a synchronous-scan-based approach to reading and writing data from/to sensors and actuators that differs fundamentally from the event-based nature of a SOA approach [23, 33]. Migration on ISA 95-Level 1 has been described to a certain extent by [38], with a focus on the transition from scan-based to SOA-event-based communication via analogue signals.

At the level of local control loops, the main benefit of applying the SOA communication infrastructure is a richer set of diagnostic and monitoring information that can be delivered and easily integrated into the SCADA systems. Using standard service protocols for the sensor and actuator data delivery, the provisioning stage can be automated to a higher degree than that possible with the current approaches. Additionally, modifications and upgrades to the system are better supported with modular and loosely coupled services and support for event-based interactions and resource discovery.

[39] describe a Service Oriented Architecture with enhanced real-time capabilities and also discuss common real-time requirements for different types of local control loops, from the communications among devices located at a production cell where the requirements are typically soft real-time constraints in the range of hundreds of ms down to control loops within a mechatronic device (such as a robot arm) with hard real-time requirements typically below 10 ms.

Implementation alternatives

As part of the IMC-AESOP project [22, 25], two main approaches were made available for migrating the existing control loops to SOA-based solutions proposed by the project:

- For control loops with low real-time requirements (loop times near 100 ms or greater), the IMC-AESOP services ‘Sensory data acquisition’ and ‘Actuator output’ can be deployed directly to the embedded sensor/actuator devices. With the use of EXI and CoAP technologies, it is possible to provide extensive and non-intrusive diagnostics and monitoring information through wireless links, and the possibilities and limitations of these technologies are discussed by [40]. In many scenarios, the achieved efficiency is envisioned to support the communication of process values via low-bandwidth wireless solutions. Legacy devices that support firmware updates can be migrated directly to this architecture. For closed black-box devices, the IMC-AESOP services ‘Gateway’ and ‘Service Mediator’ are required to provide an SOA interface and protocol mapping.
- For control loops with strict timing requirements and short loop times (less than 100 ms), the direct deployment of ‘Sensory data acquisition’ and ‘Actuator output’ requires deterministic and high-bandwidth PHY/MAC layers such as Industrial Ethernet solutions. Low-bandwidth links, e.g., (Wireless) HART, would likely require Gateway/Mediator wrapping to migrate the low-level real-time protocols used for the loops with a SOA-ready interface. Thus, the simple and time-critical sensor/actuator portions of real-time control loops are not migrated to SOA but are instead wrapped at a higher level.

A related technical aspect of the migration of local control loops is the process of physically transferring systems from legacy approaches, e.g., 4–20 mA, M-bus, and RS-232, to a service interface. Such a translation to Service Oriented Architectures solutions may include the following:

- Use of wireless solutions can provide topological and hierarchical flexibility not generally achievable using wired communication technologies. The wireless approach is slowly becoming accepted by the industry, though the academic world has long embraced it. New architectures, such as Wireless HART and 6LoWPAN, are now used in various applications. The use of wireless, and even battery-powered, devices allows for the sensing of physical properties that was not possible before either due to physical limitations (e.g., mobile sensors worn by human users) or the high installation costs with wires.
- Use of wired solutions is an option if determinism and a high performance level are required. At the physical layer, wired solutions are primarily based on the use of Ethernet cabling solutions and corresponding switches. The TCP/UDP standard communication protocols are used in these situations, together with SOA protocol stacks such as DPWS or OPC-UA [32]. In such solutions, the architectures are no longer centralized, and the intelligence is distributed among all available resources in the form of web services. However, to gain acceptance, the new SOA solution must provide a performance level comparable to that achieved using traditional field buses. Previous studies demonstrated that a DPWS stack can provide this high performance level if it uses EXI [41] instead of standard text-based XML [42].

5.2 Distributed control

As the distributed control depends on sensors and actuators from different parts of the site, it is most likely to be migrated in the fourth migration step (**Control Execution**).

Service architectures are well suited for the distribution of sensing, control and actuation over several systems or devices due to the modularity and loose coupling of service producers and consumers. Considerations regarding real-time performance similar to those made with local control loops should be made for distributed control. However, due to the nature of distributed control, the requirements are usually more relaxed.

With the less hierarchical structure of a SOA approach, the implementation of distributed control is expected to become easier, as there is less of a need to specifically implement measures to enable the transfer of data from one section of the control system to another. As more of the control execution is pushed in the devices, the last bits of a distributed control loop may be performed by other components, such as network infrastructure devices.

One other positive outcome of migrating distributed control to SOA is that a portion of the control can be temporarily disconnected without affecting the complete equipment, either for normal replacement or for upgrading functionalities. Additionally, configurations containing the control logic can be stored in a central repository such that an ex-

change of devices is possible without manual reprogramming. A Service Bus middleware component can typically support such decentralized control.

- The components of the Service Bus may be physically distributed on the following:
 - Existing devices in the system (as shown in Figure 8)
 - Existing infrastructure devices such as Gateways
 - Dedicated devices
- Certain devices contain their own local logic so that they can expose high-level services.
- Other devices cannot expose such SOA services and are either legacy devices or small devices that do not support local logic. Due to the Service Bus, these devices can nevertheless interoperate in the system.
- The remaining logic is required to ensure that the global control of the system is distributed within the Service Bus, i.e., in the example of the two devices supporting a Service Bus component.

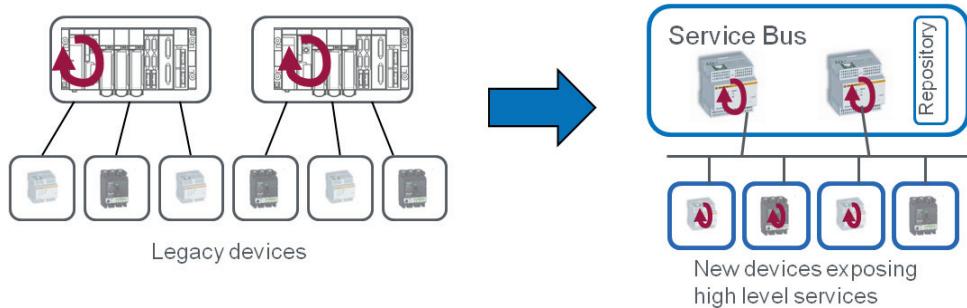


Figure 8: Distributed control with Service Bus middleware

5.3 Supervisory control

The migration of supervisory control is expected to occur in the fourth step of migration (**Control Execution**), although it may be migrated as early as the second step (**Configuration**) in some scenarios. One possibility is that new supervisory control functions are implemented in SOA as soon as possible, while the existing functionality is migrated in the third or fourth step.

In the SOA approach, devices can directly expose their data to the other systems at different levels, thus eliminating a hierarchical structure in which device data are

first collected by controllers that feed the supervisory control system. The visibility of the devices is therefore improved without additional workload. The maintenance and evolution of the supervisory application are also decoupled from other underlying systems such as controllers or OPC servers.

Supervisory control systems can also provide a richer interface, and their development is easier due to the use of tools that understand the standard interfaces exposed by the controllers and the devices. These interfaces are typically described via WSDL files.

The OPC-UA additionally provides a feature known as programming against type definitions (see Figure 9). The principle is that an OPC-UA server supports the definition of complex object types, which can be recognized by a client application, i.e., supervisory control. In the server address space, both the object type and the object instances are exposed. The supervisory control either already knows the object types exposed by the server or discovers them during the engineering phase. In both cases, the control measures applied to each object instance are programmed only once due to knowledge of the object type. In this manner, supervisory control applications can be quickly developed with libraries of components that correspond to standard object types.

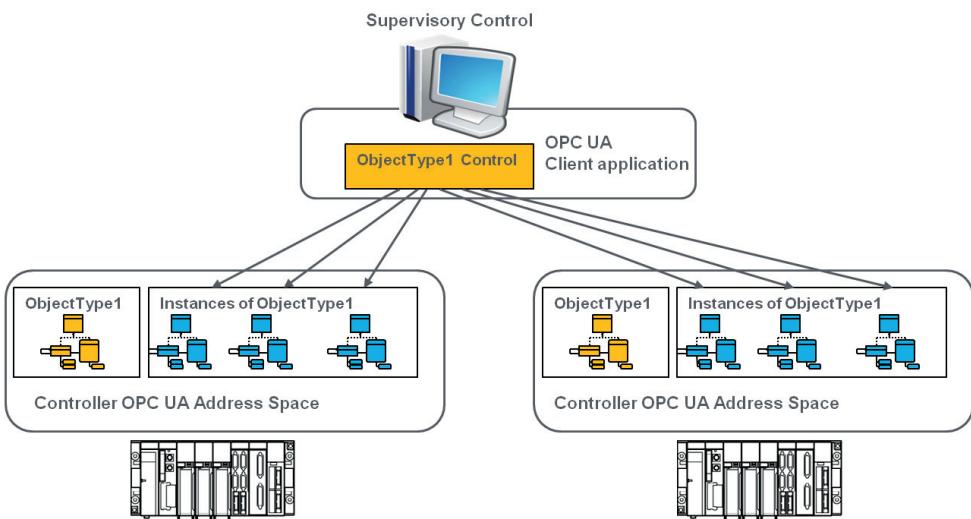


Figure 9: Programming against object types with OPC-UA

5.4 System aggregation

Because the system aggregation is intimately related to the physical situation, system architecture and production process, it can be migrated at any step depending on the

scenario.

As indicated in Figure 10, the process plants are separated into several sections. Depending on the nature of the process represented by a section, control can be realized in an encapsulated but coordinated manner via master control. This case is even more important in batch applications than in continuous processes. Batch control is a more flexible method for mastering the market demands of producing small quantities of changing products (chemical, petrochemical, medical, etc.) at the same production site. In this work, the production equipment, i.e., boiler, heat exchanger, and distillation column, is dynamically combined and controlled according to recipe needs. Support functions, such as air compression for auxiliary energy provision or cooling aggregates, are normally built as a package unit with its own controls.

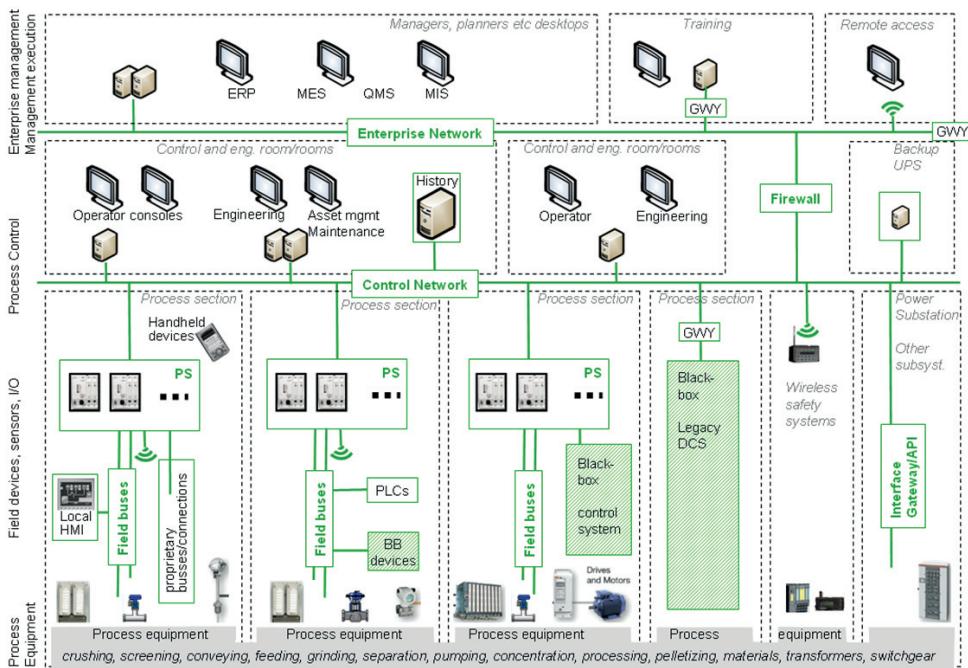


Figure 10: General architecture of a process control system

As noted, today's classical process plants and associated automation systems are already or partially characterized by the following:

- Aggregation of information dedicated to specific plant sections
- Individual engineering and control of those sections (black boxes)

- Hierarchical engineering concepts for overall/master control
- Supervision down to the black-box level

Additionally, one can begin from the process level to identify plant sections, e.g., performing individual control loops or contributing information to dedicated aspects (e.g., maintenance) of a plant view, to define data that are related to each other. Those relationships may lead to the definition of application-related services that contribute to the SOA. Certain elementary services are already defined in the IMC-AESOP architecture [37].

According to the step-by-step migration approach, these typical representations can be treated as starting points for a specific migration step that supports dedicated integration technologies. The overall migration process is a series of individual migration steps [38].

Integrating and aggregating data for this purpose require knowledge of the access path to and methods for handling data as well as the syntax and semantics of the data accessed. This information may be derived from project documentation provided by the vendor with delivery of the control equipment for a plant section or from pre-established knowledge in the case of conformance to well-established standards. Well-established standards target information management at different levels of the ISA-95 layered model that are exploitable for integration tasks:

- ISA-95
- S 88
- Device Profiles

The use of standard conforming equipment is highly recommended because this practice ensures reimbursement of investments. A summary of the concept of the aggregation of data and definition of services at Levels 1 and 2 continues to be paid attention to, as in SOCRADES [10, 43].

The Gateway and/or Mediator concept [44] was introduced for realizing integration tasks within the IMC-AESOP framework. This concept supports the representation of single resources (i.e., a legacy device) in a SOA-based environment as well as the aggregation and mediation of data from single or multiple resources.

5.5 Inter-protocol communication

Inter-protocol communication is an essential part of the migration itself and therefore must be introduced in the SOA system in the first step (**Initiation**) and should be well-integrated at the second step (**Configuration**). Modifications to this functionality are expected throughout the whole migration, as new protocols may be introduced and old protocols are gradually migrated or removed.

The interoperability of applications requires fundamental communication capabilities, even if applications are running on inhomogeneous communication platforms, which is

the usual case for integration tasks. Two or more communication channels must be mapped to each other by considering the different characteristics at all protocol layers. Different approaches known from the literature include the following:

- Bridge
- Gateway
- Router

The introduction of SOA into a process control environment necessitates integration tasks for different types of communication (i.e., 4-20 mA standard wired signals, HART protocol, fieldbus protocols such as Profibus PA and others), which all must be mapped to a the protocol agreed upon for service exchanges in the new architecture.

Within the IMC-AESOP approach, the Gateway or Mediator concepts are used for protocol mapping, covering the interfaces of different protocols, interpreting the syntax and semantics of data operated at each communication channel (possibly in a different way) and mapping data to an internal data model of the integration components. The Web Server (interface to SOA) accesses the internal data model and maps the data to an appropriate Web service with conformance to the IMC-AESOP architecture definition [37].

Configuration of this mapping requires a multi-step approach with configuration for each of the individual communication channels and the use of an internal object model that represents the targeted view of the underlying system as well as definition of the mapping roles to the Web services. Knowledge of the targeted protocols and applications is required for the implementation of the Gateway or Mediator.

More automated, transparent interoperability options have been investigated by Derhamy et al. [45, 46] and while some partial solutions are available there is still more work to be done for complete interoperability from network layer to semantics.

5.6 Data acquisition, display and storage

In the typical DCS design used in Figure 3, the functionality cuts through all layers from the data acquisition at the lowest level to servers on operator clients at the middle layer and storage on the Historian server at the top level. In terms of the migration procedure, this means that some elements are migrated in the first step (**Initiation**) and most are migrated at the third and fourth step (**Data processing** and **Control execution**).

In terms of the current state of the art, data acquisition has many possible solutions and implementations, most commonly involving a PLC or RTU connected to a fieldbus to transfer data as required. In terms of the IMC-AESOP architecture, the primary choice is to migrate to smart embedded devices capable of both acquiring the necessary data and encapsulating them in Web Services that can later be consumed by any interested party. An example of this migration can be seen in the IMC-AESOP use case called “Implementing Circulating Oil Lubrication Systems based on the IMC-AESOP Architecture” [25].

At the lowest level, this use case requires computers capable of calculating flow rates from positive displacement flow meters. These volumetric flow meters generate pulses at frequencies ranging from 1-500 Hz depending on the model chosen. Any conventional PLC or RTU has inputs that can detect a frequency of roughly 50 Hz. Although this level is sufficient for certain flow meters, it is not nearly sensitive enough to cover the entire range of possibilities. Two possible solutions to this migration problem are described:

1. One possible solution is to use a legacy flow computer with legacy communication capabilities, i.e., a modbus, which would enable the flow computer to perform the necessary high-frequency calculations and transfer the data to a modbus register that could be read by any WS-capable device. The data would subsequently be processed from pulses into a flow rate and encapsulated into a WS-Event or message depending on the requirements.
2. Another possible solution is to use a fast counter card or specialized inputs integrated into a WS-Capable device, which would imply that the device must be capable of counting, pre-processing and calculating flow rates without any external assistance. Therefore, the task would only be a matter of encapsulating the data in WS form to make them available to any interested parties.

Whichever solution is chosen, it is important to keep in mind that legacy flow metering computers, while limited, have well-defined and well-tested algorithms that calculate flow rates. In the case of migration, it is necessary to evaluate whether accuracy or scalability and easy access are more important. The common theme in both solutions, however, is the need for WS encapsulation, which would imply exactly the same work in both cases. It would be necessary to design the corresponding WSDL file such that the device that captures the information could be discovered and subscribed, although this effort might depend on how the WS-Capable device is designed to work.

5.7 Alarms and warnings

Alarms can be raised at different levels, either directly by the devices or by upper-level systems that have processed information originating from one or several sources. Therefore, the functionality can be migrated as early as in the first step (**Initiation**) in some scenarios. In most cases, however, the main part will be migrated in parallel to other data management systems in the third step (**Data processing**).

In addition to the definition of standardized interfaces that define the content of the alarms, a SOA approach proposes communication mechanisms to ensure the following:

- The right information will reach the right person in the plant with an appropriate level of detail
- The communication network of the plant will not be overcrowded by useless data

These two goals are achieved by filtering and routing mechanisms typically implemented by a Complex Event Processing (CEP) technology, such as that investigated in the IMC-AESOP Project.

For the end user, the benefit of a SOA approach is that s/he will receive only the necessary alarms and warnings. The content of the alarms will be filtered depending on the user logged into the system and will provide only the information required for the user actions. For example, an operator will be informed that the process is stopped without any further detail, whereas a maintenance team will receive details of the machine breakdown.

5.8 Emergency stop

Detection of abnormal conditions that require an emergency stop can be performed at various levels. In addition to emergency stop buttons at the shop-floor level, the events arising in the different layers of the system can indirectly inform the operators of critical alarms, typically within the supervisory control system. This distribution of functionality means that some parts of the Emergency stop will be migrated in every migration step, but the main part is likely left until the final step (**Control execution**). Moreover, complex event processing systems can correlate the information originating from different sources located in any component of the system to raise such emergency alarms.

Once an operator has physically pressed a shop-floor button or has selected the emergency stop in an HMI, the equipment must shut down in a controlled manner, which depends on the exact state and topology of the system. The agility of the SOA infrastructure allows the management of several shut-down strategies depending on the various emergency conditions as well as adaptation of these strategies along the life-cycle of the equipment.

In certain contexts, typically for regulation purposes, the shut down of the equipment must be carried out in a given time frame with a precise sequence of operations. In those cases, safety protocol solutions must be used to manage these particular constraints. Currently, different add-ons exist for classical fieldbuses, but for the envisioned systems in which an IP protocol over Ethernet is largely used, safety solutions based on Ethernet must be carefully considered.

5.9 Operator manual override

Depending on the scenario, this functionality may be partially migrated as early as in the first migration step (**Initiation**), but the main part is most likely left for migration in the third step (**Data processing**), when the operator clients are migrated along with other related systems.

The suggested implementation is for the devices to expose standardized interfaces such that the operators can use a set of tools for taking local control. Therefore, operators can be well trained and efficient, which is particularly important if an unexpected situation occurs, which is the typical case for which manual override is required.

The portions of the system in which operators have overridden the automatic control must be easily noticed in the upper-level applications, even if the event consists of a scheduled maintenance in which a portion of the system is disconnected intentionally. The SOA facilitates a direct connection between the upper level and the devices such

that critical information is easily available. The information is used not only by the operator but also by the upper-level applications for reconfiguring themselves.

Due to the loose coupling of the SOA approach, most applications at Level 2 or Level 3 will continue interacting with the manually controlled portion of the system without consideration of its operating mode. Only applications involved with the operating mode will be informed, typically via alarms and event mechanisms.

5.10 Operator configuration

Operator configuration will, just as Operator manual override, most likely be migrated in conjunction with the operator clients and related systems in the third step of the migration (**Initiation**). However, there are scenarios in which it may be migrated earlier as well.

The devices expose standardized configuration services such that a limited set of tools can be used for local configuration. Therefore, operators do not require access to many different tools and do not undergo training for all of them. The changes enacted in the device configuration must be pushed to the configuration repository so that after the replacement of a device, the same configuration can be downloaded to the new device. Different strategies can be used in this case; either the operator explicitly decides that the new settings are valid and manually initiates the backup or the device configuration may be compared periodically with the reference, which is updated if the actual device configuration is different but valid.

Figure 11 describes a system in which the standard service DeviceManagement is supported by an IMC-AESOP device. A local configuration tool can be used to perform the following actions:

1. The current configuration of the device is obtained. The response of the GetConfiguration operation is defined in a generic format, and virtually any type of device configuration can be retrieved.
2. The operator edits the device configuration with the configuration tool HMI.
3. The tool uploads the new configuration in the device (SetConfiguration operation).
4. As an option, the new configuration is pushed into the configuration repository. This repository will be used in case of device replacement.

5.11 User management and security

Till recently, a common procedure used locally authenticated users (e.g., on-device or in-department) and devices (if at all). However, this practice created ‘islands’ within the infrastructures that were difficult to control, e.g., adherence to corporate policies and cost of maintenance. In the IMC-AESOP vision, the security framework should be implemented company-wide, and the ‘visibility’ of devices makes it easier to gain

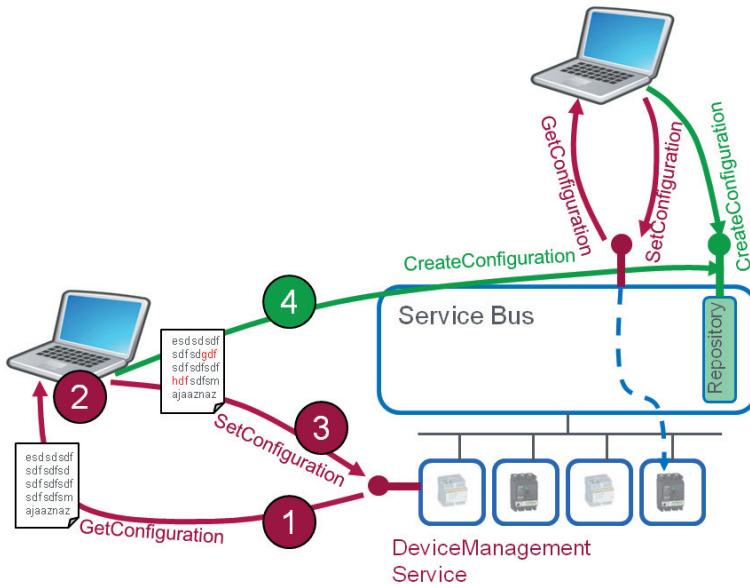


Figure 11: Operator local configuration with a Service Bus

a system-wide view. However, the migration towards this infrastructure will require a lengthy transition process and potentially significant efforts to reassess the relevant security and risk aspects, test configurations and impacts and move towards integrated management of both users and their rules. To ensure a secure and manageable migration, this aspect must be taken into account throughout the whole migration process.

6 A first step - Migration of a lubrication system

In this section, the authors detail the development, implementation and evaluation of a demonstrator on the premises of LKAB in Sweden (please refer to Figure 12) to present the first step of the migration procedure and highlight some of the challenges in migrating an existing plant lubrication system from a classical control system to the new approaches addressed by the IMC-AESOP project. This demonstrator was part of the IMC-AESOP project and was first presented by [47].

Lubrication systems are typical critical systems for almost all process industries. The lubrication control system provides important information that can be used by operators to avoid critical and damaging incidents by operators, by planning staff and management to improve production and plant efficiency and by maintenance staff and management to analyse and improve predictive maintenance.

The plant lubrication use case addresses a number of key points, i.e., migration from a scan-based PLC to an event-based SOA system, enabling the SOA on low-level devices



Figure 12: The LKAB pelletizing plant KK4 in Kiruna, Sweden

and in closed-loop control and integration into an actual plant environment. [23] discuss the advantages of using SOA-based solutions for industrial process monitoring and control.

The lubrication system presented in Figure 13 is deployed at the LKAB pelletizing plant [48] as one of a number of independent black-box systems supporting the main process. Such systems are characterized by limited data exchanges between the lubrication systems and the larger distributed control system (DCS). This lubrication system was migrated from the current implementation using a PLC to a SOA system.

6.1 Existing system

As shown in Figure 13, the existing lubrication system includes two lubrication circuits controlled by a PLC (Programmable Logic Controller) that receives start/stop commands from a DCS. Each lubrication circuit is connected to a pump controlled by the DCS through a digital output. More than 70 AS-i [49] position switches combined with various digital inputs are scanned periodically by the PLC to obtain fluid distribution status updates for each lubrication circuit. Based on the sensor information, the PLC controls each pump and directs the fluid to the appropriate circuit. As mentioned previously, rather limited communication occurs within the operational layer, although a touch panel

provides local supervision capability.

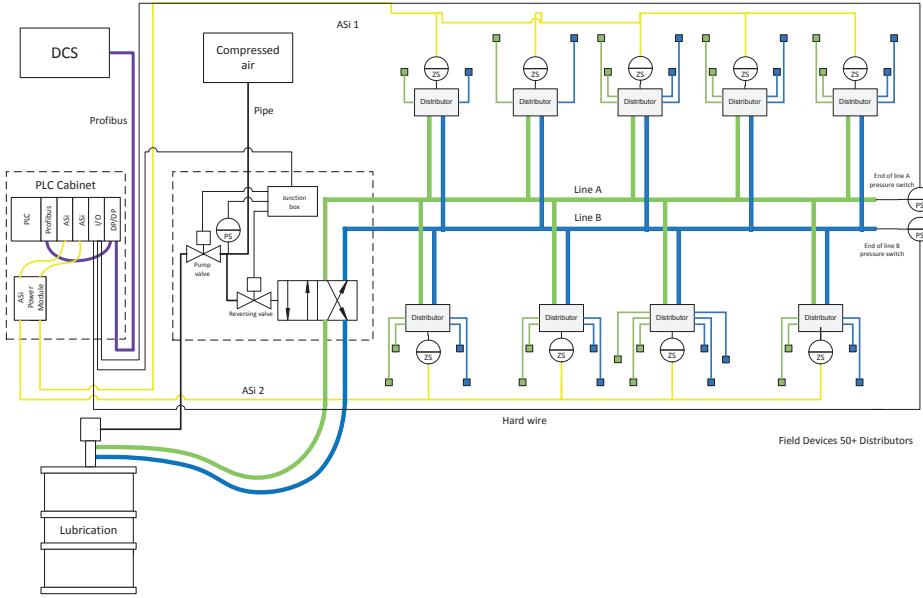


Figure 13: Existing system

6.2 Demonstrator setup

The prototype used for the demonstration consists of the replacement of the existing PLC with a SOA-based system. Thus, the current PLC cabinet is replaced with a SOA-based cabinet and connected to a maintenance station, as shown in Figure 14. As a precaution during the migration, a multiplexer was installed to allow for quickly switching back to the PLC cabinet if any issues arose with the SOA installation.

6.3 SOA components

The proposed SOA architecture is illustrated in Figure 15. Only the DCS portion (dotted line) and the AS-i slaves are inherited from the legacy system; other components (including the SCADA) belong to the SOA demonstrator. From top to bottom, we have the following:

- The SCADA provides advanced local control and monitoring capabilities.

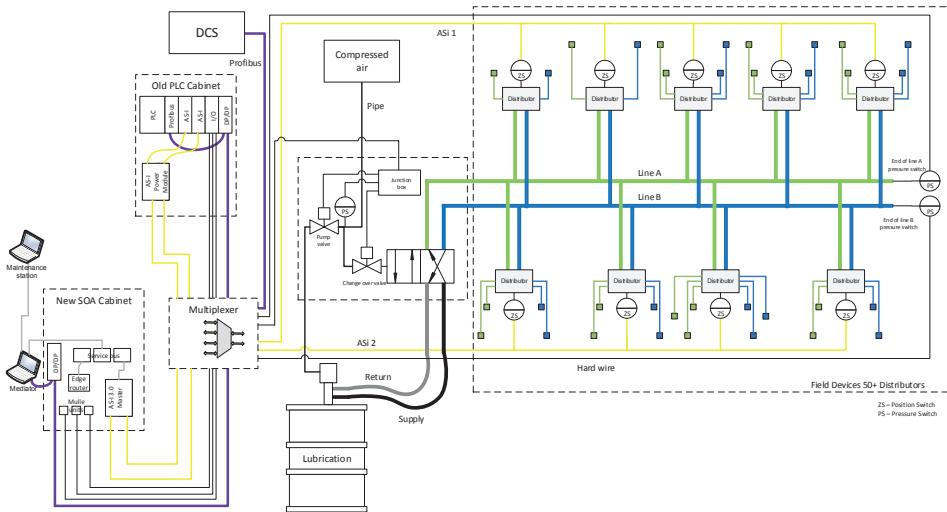


Figure 14: Proposed Prototype

- The Mediator provides an interface between the device layer (with the DPWS protocol) and the control and supervision layers (with both Profibus and OPC UA protocols).
- The Service Bus provides an abstract layer on top of the field devices, providing both synchronous and asynchronous data collection mechanisms; it also implements the control logic that was originally run from the PLC.
- Wireless sensors and actuators (Mulle nodes and edge router) provide process I/O facilities as services.

SCADA

To replace and extend the HMI functionality provided in the legacy system by an integrated touch panel connected to the PLC, a commercially available SCADA solution was selected and configured for this use case. The configuration of the SCADA solution mainly consisted of selecting which parameters to display, how to display them, and which actions would be allowed through the user interface. For this demonstration the user interface was designed to be similar to the replaced HMI and to allow similar functionality, to accommodate operators familiar with the old interface. This solution provides a flexible method of presenting data and configuring the system parameters.

Using an OPC UA client that accesses the server provided by the Mediator, the system can be accessed from anywhere on the connected network rather than by current local

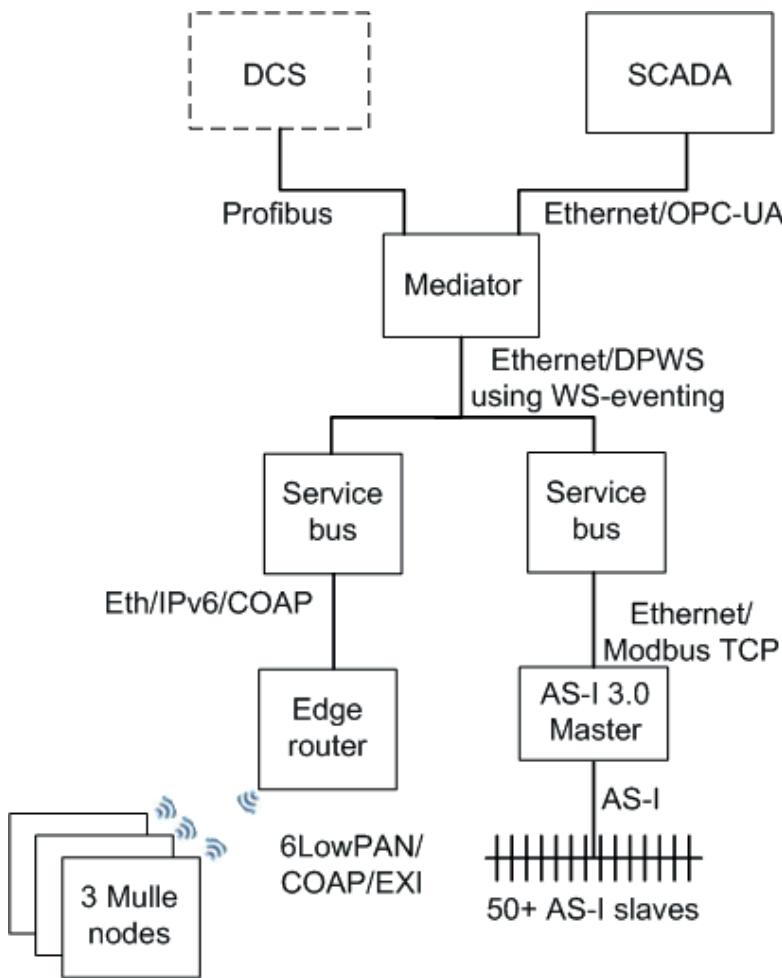


Figure 15: Proposed prototype of Service Oriented Architecture

access only. At the same time, the OPC UA server provides a flexible method to access the system with other standardized tools that provides a wide array of possibilities.

Mediator

The Mediator provides a runtime system for monitoring and control of process facilities by integrating both legacy and SOA-based technologies. The core section consists of a data model that describes the logical view of the monitored facilities and also contains all relevant information for acquiring the data, including communication. The Medi-

ator communicates with the Service Bus through the DPWS and also supports basic authentication over SOAP.

For the integration of different communication protocols and information models of various devices and other data sources, an abstract data access layer is introduced. For the application described in this paper, the PROFIBUS protocol (for connecting to the DCS) and the DPWS protocol (for connecting to the Service Bus) have been implemented. Similarly, any processing of data for pre-processing, control, KPI calculation or presentation to the SCADA HMI layer is easily extendible by providing the appropriate adapters.

Within the framework of the SOA system described above, the Mediator data model (including alarms) is presented to the HMI of the maintenance application using the OPC UA protocol.

Distributed Service Bus (DSB)

As a complement to the Mediator, the Distributed Service Bus provides an additional integration of heterogeneous systems that support various communication media, protocols, and data models, as shown in Figure 16. Such integration is enabled via loose coupling-based protocol connectors. Each protocol connector reifies the devices and services of an existing system into the DSB data model representation. Thus, using a defined abstract layer, the Service Bus provides a common representation of those devices and services. This abstract layer enables a wide variety of common operations on the underlying systems and devices, including management, diagnostic, maintenance and monitoring.

Moreover, the distribution of the Service Bus provides scalability and evolution because each instance can be configured for a specific application domain by implementing dedicated interfaces, quality of service and security requirements. Devices and services handled by an instance of the Service Bus are reified, and their information is shared between the other instances through the DPWS protocol.

The distribution feature provided by the DSB is particularly suited to the management of large-scale distributed systems, which is central to IMC-AESOP and this demonstrator in particular.

The Service Bus was implemented on two Raspberry Pi devices running the Linux operating system and featuring 512 MB of RAM and 700-MHz ARM CPUs. As illustrated in Figure 16, the main software components of the Service Bus are a pivot data format, a set of connectors acting as external interfaces (DPWS, REST, CoAP, and Modbus), an event module, a time synchronization (PTP) module, a logging (syslog) module and the AESOP logic, which reproduces the application logic from the existing PLC.

The two instances of the Service Bus dynamically discover each other at startup using WS-Discovery and rely on DPWS for message exchanges. Basic cyber-security protection is provided by the combination of Role Base Access Control (RBAC) and a user authentication mechanism.

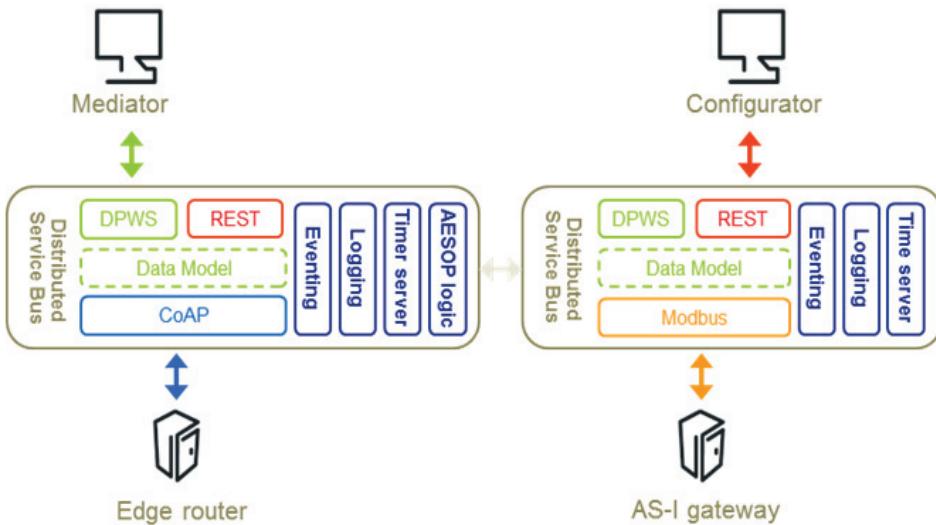


Figure 16: DSB architecture

Wireless sensor and actuator network (WSAN)

Mulle devices [50] serve as I/O nodes that connect lubrication pressure switches, air pressure switches, pump valves, reversing valves, and indication lights. To render the system scalable and integrate it with the IMC-AESOP service cloud, IPv6 was chosen as the network protocol. To ensure that the IPv6 network layer complies with the IEEE 802.15.4 Link layer, the 6LoWPAN adaptation layer is used, and 6LoWPAN compresses and reduces the data overhead such that less energy is required to transfer the information between wireless nodes.

Mulle nodes communicate using the Efficient XML Interchange (EXI) (www.w3.org/XML/EXI) and CoAP on top of 6LoWPAN. The services hosted by the Mulle support input, output, filtering, logging, and configuration services. All data are EXI encoded and transmitted using CoAP over 6LowPAN.

The edge router performs translation between the IPv4 over Ethernet and IPv6 over 6LoWPAN (IEEE 802.15.4) networks. The edge router also hosts time synchronization services (NTP and PTP) and CoAP services, such as data proxy, and logs the performance of the WSAN. The CoAP is a protocol designed for scalability and simplicity [51] and is backwards compatible with the frequently used HTTP protocol.

Representation of the information measured by the sensors in an efficient yet self-explanatory manner is desirable. Because the bandwidth in the wireless sensor network is limited and the energy available in each sensor node is also limited, the efficiency parameter requires extra attention. The concept of Service Oriented Architecture (SOA) is highly interesting in this context because each measured parameter can be represented

as a service to the other nodes but is also represented globally because the sensors are connected to the internet using IPv6.

In the demonstration setup, a total of 14 CoAP services (4 actuators, 6 sensors, and 4 outputs used to indicate the system status) were implemented. These components were located on three nodes, each running on Mulle v6.2 from Eistec [50].

A key component of the migration of legacy systems to SOA is the use of standard and globally accepted formats for representing the exchanged information. One important result of this demonstration is that it is possible to use EXI for the integration of sensor and actuator devices with the SOA automation infrastructure. This method enables the implementation of RESTful web services based on CoAP and EXI for industrial applications with low real-time requirements.

6.4 Data modelling

Enabling the interoperability of the service specifications and data models is a key technological challenge that SOA systems are intended to resolve. Full interoperability requires that the syntax and semantic service descriptions are well defined, unambiguous and enable dynamic discovery and composition. Thus far, most if not all SOA installations enable pure syntax interoperability with little or no support for standard-based semantic descriptions. The use of structured data formats only partially resolves the problem by supplementing the exchanged data with meta-information in the form of tags and attributes such as in the case of XML/EXI. The tag names are ambiguous and usually insufficient to describe the service functionality in full.

The use of application-level data model standards is often used as a solution to this problem because the syntax to semantics mapping is predefined. An example of such a standard is the Smart Energy Profile 2 that clearly states the physical meaning of the tag names and structures defined for the service messages in the domain of energy management. One problem in compliance with such standards is that they are almost always domain specific, which requires mapping of the semantic descriptions from one standard to all others in use.

Another approach defines a generic semantic data model that is applicable to a wide range of use cases. This approach is selected for the work presented in this paper. The initial investigation highlighted the Sensor Model Language (SensorML) [52] developed by the Open Geospatial Consortium (OGC) as a promising specification for generic semantic description of sensory data. However, the complexity and size of SensorML specifications limit its use to more capable devices. Small-scale experiments with a number of sample SensorML messages showed that even the EXI representation will not be sufficiently small to fit in battery-powered wireless sensor nodes that contain low-power/low-bandwidth radios.

Another possible specification for sensor data is the Sensor Markup Language (SenML) [53], which has a rather simple design that is consistent with RESTful architecture and is targeted to resource-constrained devices. The evaluation of SenML specifications shows that it meets the requirements for hardware utilization, but there are areas that are too

highly simplified and insufficient to describe the data with the level of detail required by the target application. An example of such a limitation is the precision of the timestamping for sensor data; SenML allows for up to seconds of resolution, which is not sufficient for most use cases. This situation led to the use of a custom generic data representation that reuses many of the design choices in SenML.

6.5 Migration aspects

The overall prototype was installed in a cabinet similar to that installed in the plant in order to facilitate the on-site temporary installation of the prototype. As discussed earlier in this paper, migration of a large DCS into the SOA can be initiated with a smaller step in which certain key functionalities are migrated, and the basis of the SOA infrastructure is established in a subset of the plant.

This use case provides an example of the migration of a number of functional aspects that have been identified in the existing system and also provides a minimum requirement of functionality for the SOA-enabled system. The most significant of these aspects are described in the following subsections.

Local control loop

In the existing system, local control is performed within the PLC using the internal timers and the pressure switches distributed throughout the system to trigger the start/stop of the lubrication pump and activation of the solenoid valves.

In the IMC-AESOP use case, this functionality is distributed primarily to the Service Bus, which accesses both the CoAP services provided by the Mulle nodes for sensing and actuating the AS-i sensor data. The main advantage of the SOA design is that it provides added monitoring capabilities for the control loop (the timers and sensor data are available as services).

Inter-protocol communication

In the existing system, only two communication protocols are involved. The communication to the DCS is handled through Profibus, and the collection of data from a large number of field devices is handled through AS-i.

In the demonstrator, several new protocols are introduced in the architecture to allow communication within the SOA system, whereas the existing communication interfaces remain accessible through the commercially available AS-i and Profibus master modules. The conversions between different protocols are handled by the Service Bus and the Mediator, as previously described.

Alarms and warning

In the existing system, alarms are handled via lists of Fault- and Reset-bits with a corresponding list of alarm texts, which are both controlled in the PLC.

In the SOA solution, those alarms are implemented as events collected from the alarm sources and brokered by the Service Bus. Any interested party can subscribe to these alarms from the Service Bus. In the demonstrator, the SCADA, the DCS (both through the Mediator) and the Service Bus web client are subscribers of the process-level alarms. Polling-based alarms remain available, which is particularly interesting in a migration context.

Operator manual override and Operator configuration

Operator manual override and Operator configuration are the two key functionalities provided by the touch panel HMI in the existing system.

In the SOA alternative, the Service Bus exposes these two functionalities as services that can be called by any (authenticated and authorized) client application. In the demonstrator, two client applications consume these services: the SCADA (through the Mediator) and the Service Bus web client.

As mentioned previously, the loose coupling provided by this approach can be leveraged in future maintenance operations by allowing transparent and independent replacement either by the server or the client of those services.

6.6 Functional assessment

The functionality to be delivered was based on an analysis of a backup copy of the code running on the existing PLC, which was written by Midroc and confirmed with the LKAB technicians. This analysis was used both to construct the logic implemented in the DSB and to write a protocol for verifying the functionality. The test procedures and the protocol were first used in the initial integration of components in which the components were first brought together and connected in a local network.

The test procedure was based on the common industrial practice employed by Midroc in most industrial projects, and Midroc templates were used for the test protocols. The procedure consists of a series of acceptance tests beginning with the Internal Acceptance Test (IAT), which is performed by the engineers integrating the system and without any customer representatives. Once the IAT has been passed with acceptable performance, the customer is invited to the system integrator site for a Factory Acceptance Test (FAT) to verify that the functionality provided is in accordance with the customer expectations. To finally verify that the functionality is accurate, a Site Acceptance Test (SAT) is performed at the site after commissioning.

This common practice was used during the development of the demonstrator, and as the system was developed, the test procedure was extended at several points. The final version of the test procedure and protocol was used in the FAT at the Midroc facilities in Stockholm, Sweden, with representatives from LKAB acting as the customer to verify that the functionality and reliability were satisfactory for use in the process plant. Figure 17 shows one page of the 18-page FAT protocol describing the prerequisites for the tests and step-wise instructions for 11 scenarios with the expected results at each step.



5 Test descriptions

5.1 Scenario 1, Normal operation GSS

5.1.1 Purpose

Scenario 1 describes the lubrication procedure of Grate Support Shaft during normal operation when no faults occur.

5.1.2 Special conditions

During these tests push-buttons and induction switches will be used instead of the position and pressure switches used in the actual application. LED's will be used to indicate the state of the actuators instead of the pumps and valves used in the actual application.

Switch "End-of-Line pressure" is manually switched within predetermined time-out, otherwise this will be scenario 5.

A Siemens S7 PLC will be used to act as the DCS connected to the DP/DP-coupler.

5.1.3 Exceptions

No exceptions planned.

5.1.4 Test procedure

The procedure is performed in the order of the listed steps. Since this scenario is supposed to be automatic most steps are triggered by one of the proceeding steps. The exception to this being where the next step would be triggered by the physical lubrication system, such as the End-of-Line pressure switch.

If the expected result is achieved it is checked in the list using the initials of the testing personnel. If not, a note of the deviation is made in the checklist.

No	Step	Procedure	Expected results	Checked
1.1.A	Normal start	Signal "Start GSS" (0x0100) is set from the DCS through the DP/DP-coupler	"System GSS On" is acknowledged to the DCS	
1.1.B			Indication light "System On" flashing at 1 Hz	
1.1.C			System GSS ON is indicated on the Maintenance Station	
1.1.D			"Pause timer GSS" status is indicated on the Maintenance Station	
1.2.A	Auto start pump	Triggered when the pause timer is finished	"Pause timer GSS" is Re-started and its status is indicated on the Maintenance Station	

Figure 17: Extract from the FAT protocol

The final functional validation of the overall architecture was performed on-site during a scheduled plant maintenance break. The IMC-AESOP prototype was connected to the lubrication system by disconnecting the operating cabinet and connecting the SOA cabinet.

The lubrication system was run for several hours to validate the functional behaviour of the prototype and to collect timing data.

6.7 Performance assessment

To measure the overall performance of the prototype, the components of the Service Oriented Architecture synchronized their timing using the PTP protocol (IEEE 1588). All of the components were configured to send their logs to a centralized syslog server (IETF RFC 5424) for timing analysis. Table 1 summarizes the average time it took for an End-of-line pressure switch event to propagate from the Mulle device to the Mediator through the Edge Router and the Service Bus.

Table 1: Time Measurements

Event	Node	Time offset (ms)
End-of-line pressure switch	Mulle (sensor)	0
	Edge Router	11
	Service Bus	13
	Mediator	21

In this example, the CoAP Edge Router receives the event 11 ms after the Mulle sensor detects the end-of-line pressure switch, the Service Bus acknowledges the event 2 ms later, and finally, the Mediator receives it 8 ms later. The total transmission time between the sensor (Mulle) and the Mediator is 21 ms, which is longer than the current PLC cycle time but compatible with the application requirements.

6.8 Wireless assessment

One parameter of interest that is important for successful deployment of 6LoWPAN devices is the size of the messages that the devices must exchange. Use of XML is beneficial for the integration of the devices with the data models and the message formats used in the upper layers of the automation. Using EXI in strict XML schema mode for the low-bandwidth wireless links, the size of the XML messages is reduced by a factor of more than 20. Therefore, the size of an EXI-encoded digital IO process value with a timestamp and quality indicator is 10 bytes compared with 228 bytes for its plain XML counterpart. Another key performance indicator for wireless applications, especially in noisy industrial environments, is the occurrence of retransmissions of packets. A retransmission wastes bandwidth, uses additional energy and increases latency. During the tests, retransmissions were observed at a low level with a stable wireless network as a result.

6.9 Overall drawbacks and benefits

A general drawback of the proposed solution is obviously its lack of maturity, in the sense that it consists of a set of prototypes provided by different partners, with none of them yet industrialized. This situation translates into both unreliability issues and integration complexity. A portion of the integration difficulties consists of the requirement for a specific configuration and monitoring interface for each partner component.

This heterogeneity and relative complexity in the demonstrator can in turn be perceived as an opportunity to validate the SOA approach, with each component of the architecture exposing and consuming services to/from other components with a fairly high level of loose coupling.

In a productized version of the demonstrator, all middleware components (Mediator, Service Bus, Edge Router and potential AS-I Gateway) would ideally be merged into one product, thus reducing the main complexity of the system. However, a projection to a productized version of the SOA middleware would still lead to a higher level of internal complexity compared with that of a less versatile PLC-based solution.

The main benefit of the proposed solution compared with the installed solution is facilitation of the overall system installation and maintenance. Although the installation benefit was not obvious in the demonstrator due to the multiplicity of technologies (and partners) involved, the maintenance and monitoring value was fairly obvious due to the advanced monitoring capabilities provided by the added services and displayed through the SCADA (timers, sensors values, alarms).

7 Conclusion

Delving deeper into the migration concepts first introduced and detailed by [38, 28], the strategy for migrating from a ISA'95-based legacy process control system to the SOA involves proceeding in a structured manner by gradually upgrading the highly integrated and vendor-locked standards to a more open structure while maintaining functionality. A procedure used to migrate the functionality of a DCS/SCADA to a SOA-based implementation is proposed. The procedure is composed of four distinct steps and makes use of mediator technology. These four steps are designed to maintain the feeling of conformity between HMI and control execution and to ensure that the target system exhibits full transparency and supports open standards. It is important that the initiation in Step 1 considers these issues to enhance the plug-and-play features of the SOA system even in an industrial process control system. The portions of the DCS/SCADA in which operations and engineering are handled are migrated in Step 2 and Step 3 in a structured manner. At the fourth step, the migration of control execution, the migration approach is decided based on functionality and more detailed performance requirements. If the fade-out approach is used, the most critical control loops and control logic might stay with the legacy set up in which the operational, engineering and maintenance staff has confidence.

Using this step-wise approach with SOA and mediator technology, it is argued that the

SOA approach will preserve functional integration, support grouping of devices, preserve real-time control and successfully address safety loops. Emphasizing the DCS portion of an example legacy SCADA/DCS, the authors applied the approach and presented the results obtained throughout the entire migration process.

Within the scope of the IMC-AESOP project [22], the first step of the migration was implemented with prototype technology from several partners in a use case at the LKAB pelletizing plant in Kiruna, Sweden. The on-site validation of the IMC-AESOP prototype provided positive feedback, considering that both functional and performance results were in line with customer expectations and were combined with added supervision and control capabilities at the SCADA level.

The SOA proved to be valuable both at the device and application levels by providing a high level of loose coupling between the various components of the system. Events complemented the SOA architecture quite well by reducing the overall latency of the information flow.

On the wireless side, the tests show that CoAP-based services over 6lowPAN can be used for process monitoring and control applications with no low-latency requirements. However, additional research is required to improve both scalability and robustness and to minimize latency.

One of the major results of the migration is the transformation of the ISA-95 architecture into an Automation Service Cloud. Future work will focus on extending the migration process and procedures to comprise the full ISA-95 architecture, including security issues.

Acknowledgment

The authors are grateful for support from the European Commission and thank the partners of the EU FP7 project IMC-AESOP (www.imc-aesop.eu) for fruitful discussions. Additional support was provided by the ARTEMIS Joint Undertaking and the partners of the Arrowhead project (www.arrowhead.eu).

The authors also thank the LKAB company for open and collaborative work on this solution and for providing access to their Kiruna facilities.

References

- [1] P. Lingman, J. Gustafsson, A. O. Johansson, O. Ventä, M. Vilkko, S. Saari, J. Tornberg, and A. Siimes, “European Roadmap for Industrial Process Automation,” ProcessIT.eu, Tech. Rep., 2013. [Online]. Available: <http://www.processit.eu/roadmap>
- [2] T. Erl, *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. The Prentice Hal, 2005.

- [3] N. Komoda, "Service oriented architecture (soa) in industrial systems," in *Industrial Informatics, 2006 IEEE International Conference on.* IEEE, Aug 2006, pp. 1–5.
- [4] M. Jamshidi, Ed., *Systems of Systems Engineering: Principles and Applications.* CRC Press, Nov. 2008.
- [5] S. Simanta, E. Morris, G. Lewis, and D. Smith, "Engineering lessons for systems of systems learned from service-oriented systems," in *4th Annual IEEE Systems Conference, 5-8 April, San Diego, CA.* IEEE, 2010.
- [6] IEC62264-3, "Iec 62264-3, enterprise-control system integration - part 3: Activity models of manufacturing operations management," IEC, Tech. Rep., 2007.
- [7] H. Bohn, A. Bobek, and F. Golatowski, "Sirena - service infrastructure for real-time embedded networked devices: A service oriented framework for different domains," in *Proc. of the International Conference on Networking, Systems, Mobile Communications and Learning Technologies,* IEEE Computer Society. IEEE Computer Society, 2006.
- [8] S. de Deugd, R. Carroll, K. E. Kelly, B. Millett, and J. Ricker, "Soda: Service oriented device architecture," *Pervasive Computing, IEEE*, vol. 5, no. 3, pp. 94–96, july-sept. 2006.
- [9] V. Trifa, D. Guinard, and M. Koehler, "Messaging methods in a service-oriented architecture for industrial automation systems," in *Networked Sensing Systems, 2008. INSS 2008. 5th International Conference on.* IEEE, June 2008, pp. 35–38.
- [10] A. W. Colombo and S. Karnouskos, "Towards the Factory of the Future: A Service-oriented Cross-layer Infrastructure," in *ICT Shaping the World: A Scientific View,* J. Wiley and Sons, Eds. European Telecommunications Standards Institute (ETSI), John Wiley and Sons, 2009, vol. 65-81.
- [11] A. W. Colombo, S. Karnouskos, and J. M. Mendes, "Factory of the Future: A Service-Oriented System of Modular, Dynamic Reconfigurable and Collaborative Systems," in *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management,* L. Benyoucef and B. Grabot, Eds. Springer, 2010.
- [12] S. Karnouskos, D. Savio, P. Spiess, D. Guinard, V. Trifa, and O. Baecker, "Real World Service Interaction with Enterprise Systems in Dynamic Manufacturing Environments," in *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management,* L. Benyoucef and B. Grabot, Eds. Springer, 2010.
- [13] S. Feldhorst, S. Libert, M. ten Hompel, and H. Krumm, "Integration of a legacy automation system into a soa for devices," in *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on.* IEEE, Sept 2009, pp. 1–8.

- [14] R. Kyusakov, J. Eliasson, J. Delsing, J. van Deventer, and J. Gustafsson, "Integration of wireless sensor and actuator nodes with it infrastructure using service-oriented architecture," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 43–51, Feb 2013.
- [15] PLANTCockpit, "Production logistics and sustainability cockpit," 2013, <http://www.plantcockpit.eu>.
- [16] A. Dennert, A. Gossling, J. Krause, M. Wollschlaeger, and A. Henao Montoya, "Vertical data integration in automation based on iec 61499," in *Factory Communication Systems (WFCS), 2012 9th IEEE International Workshop on*. IEEE, May 2012, pp. 99–102.
- [17] A. Dennert, J. A. Garcia Izaguirre, J. Krause, S. Hesse, J. L. Martinez Lastra, and M. Wollschlaeger, "Advanced concepts for flexible data integration in heterogeneous production environments," in *Proceedings of the 11th IFAC Workshop on Intelligent Manufacturing System, IMS 2013, Sao Paulo, Brazil*. IFAC, 2013.
- [18] KAP, "Knowledge, awareness, and prediction of man, machine, material, and method in manufacturing," 2014, <http://www.kap-project.eu>. [Online]. Available: <http://www.kap-project.eu>
- [19] T. Palpanas, "Real-time data analytics in sensor networks," in *Managing and Mining Sensor Data*, C. C. Aggarwal, Ed. Springer US, 2013, pp. 173–210. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-6309-2_7
- [20] A. Majumdar and H. Szigeti, "ICT FOR MANUFACTURING - The Action-PlanT Vision for Manufacturing 2.0," 2011, <https://setis.ec.europa.eu/energy-research/sites/default/files/static-projects/files/vision.pdf>.
- [21] ARTEMIS, "Advanced research & technology for embedded intelligence and systems," 2011. [Online]. Available: <https://artemis-ia.eu/>
- [22] IMC-AESOP, "Aesop - architecture for service oriented process monitoring and control," 2012, <http://www.imc-aesop.eu>. [Online]. Available: <http://www.imc-aesop.eu>
- [23] S. Karnouskos, A. W. Colombo, F. Jammes, J. Delsing, and T. Bangemann, "Towards an architecture for service-oriented process monitoring and control," in *Proceedings IECON 2011*. IEEE Industrial Electronics Society, 2011, p. 6.
- [24] S. Karnouskos and A. W. Colombo, "Architecting the next generation of service-based scada/dcs system of systems," in *Proceedings IECON 2011*. Melbourne: IEEE, Nov. 2011, p. 6.
- [25] A. 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*. Springer Publishing Company, Incorporated, 2014.

- [26] A. W. Colombo, T. Bangemann, and S. Karnouskos, “Imc-aesop outcomes: Paving the way to collaborative manufacturing systems,” in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, July 2014, pp. 255–260.
- [27] J. Morgan and G. E. O’Donnell, “Enabling a ubiquitous and cloud manufacturing foundation with field-level service-oriented architecture,” *International Journal of Computer Integrated Manufacturing*, vol. 0, no. 0, pp. 1–17, 2015. [Online]. Available: <http://dx.doi.org/10.1080/0951192X.2015.1032355>
- [28] J. Delsing, F. Rosenqvist, O. Carlsson, A. Colombo, and T. Bangemann, “Migration of industrial process control systems into service oriented architecture,” in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*. IEEE, Oct 2012, pp. 5786–5792.
- [29] Z. Xu, H. Zhao, S. L. Tan, and L. Liu, “Service-driven migrating of enterprise information systems: A case study,” in *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, vol. 1. IEEE, Dec 2012, pp. 127–136.
- [30] R. Khadka, A. Idu, J. Hage, and S. Jansen, “Legacy to soa evolution: A systematic literature review,” *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments: Challenges in Service Oriented Architecture and Cloud Computing Environments*, p. 40, 2012.
- [31] A. Kalogeras, J. Gialelis, C. Alexakos, M. Georgoudakis, and S. Koubias, “Vertical integration of enterprise industrial systems utilizing web services,” *Industrial Informatics, IEEE Transactions on*, vol. 2, no. 2, pp. 120–128, May 2006.
- [32] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer, 2009.
- [33] F. Jammes and H. Smit, “Service-oriented paradigms in industrial automation,” *Industrial Informatics, IEEE Transactions on*, vol. 1, no. 1, pp. 62 – 70, feb. 2005.
- [34] S. Karnouskos, T. Bangemann, and C. Diedrich, “Integration of legacy devices in the future soa-based factory,” *{IFAC} Proceedings Volumes*, vol. 42, no. 4, pp. 2113 – 2118, 2009, 13th {IFAC} Symposium on Information Control Problems in Manufacturing. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1474667016341155>
- [35] SOCRADES, “Socrades: A web service based shop floor integration infrastructure,” 2009, <http://www.socrades.net>. [Online]. Available: <http://www.socrades.net>
- [36] C. Héault, G. Thomas, and P. Lalanda, ““mediation and enterprise service bus: a position paper” in first international workshop on mediation in semantic web services,” in *Proc. MEDIATE 2005*. CEUR, 2005.

- [37] S. Karnouskos, A. W. Colombo, K. Manninen, R. Camp, M. Tilly, T. Bangemann, P. Stluka, F. Jammes, J. Delsing, and J. Eliasson, “A soa-based architecture for empowering future collaborative cloud-based industrial automation,” in *Proc IEEE IECON 2012*. Montreal, Canada: IEEE, Oct. 2012.
- [38] J. Delsing, J. Eliasson, R. Kyusakov, A. Colombo, F. Jammes, J. Nessaether, S. Karnouskos, and C. Diedrich, “A migration approach towards a soa-based next generation process control and monitoring,” in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*. IEEE, Nov 2011, pp. 4472–4477.
- [39] T. Cucinotta, A. Mancina, G. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, “A real-time service-oriented architecture for industrial automation,” *Industrial Informatics, IEEE Transactions on*, vol. 5, no. 3, pp. 267–277, Aug 2009.
- [40] G. Moritz, F. Golatowski, C. Lerche, and D. Timmermann, “Beyond 6lowpan: Web services in wireless sensor networks,” *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 4, pp. 1795–1805, Nov 2013.
- [41] A. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, “Web Services for the Internet of Things through CoAP and EXI,” in *Communications Workshops (ICC), 2011 IEEE International Conference on*. IEEE, 2011.
- [42] R. Kyusakov, J. Eliasson, and J. Delsing, “Efficient structured data processing for web service enabled shop floor devices,” in *Industrial Electronics (ISIE), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 1716–1721.
- [43] M. Taisch, A. W. Colombo, S. Karnouskos, and A. Cannata, “SOCRADES Roadmap: The future of SOA-based factory Automation,” Dec. 2009.
- [44] S. Karnouskos, T. Bangemann, and C. Diedrich, “Integration of Legacy Devices in the Future SOA-based Factory,” in *13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM), Moscow, Russia*. IFAC, 3-5 June 2009.
- [45] H. Derhamy, J. Eliasson, and J. Delsing, “Iot interoperability - on-demand and low latency transparent multi-protocol translator,” *IEEE Transactions on Services Computing*, 2016.
- [46] H. Derhamy, “Towards interoperable industrial internet of things - an on-demand multi-protocol translator service,” Licentiate thesis, LTU, 2016.
- [47] P. Nappey, C. El Kaed, A. Colombo, J. Eliasson, A. Kruglyak, R. Kyusakov, C. Hubner, T. Bangemann, and O. Carlsson, “Migration of a legacy plant lubrication system to soa,” in *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*. IEEE, Nov 2013, pp. 7440–7445.

- [48] LKAB, "Lkab kiruna, home to the world's largest underground iron ore mine," Luossavaara-Kiirunavaara AB, 2014, luossavaara-Kiirunavaara Aktiebolag. [Online]. Available: <http://www.lkab.com/en/About-us/Overview/Operations-Areas/Kiruna/>
- [49] AS-International, "Asi complete AS-interface specification version 2.1, supplement/amendment," AS-International, Frankfurt am Main, Tech. Rep., 2002, <http://www.as-interface.com>.
- [50] EISTEC, "Eistec," 2010, <http://www.eistec.se>. [Online]. Available: <http://www.eistec.se>
- [51] K. Kim, S.-W. Lee, D. geun Park, and B.-C. Lee, "Ptp interworking 802.15.4 using 6lowpan," in *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, vol. 01. ICACT, Feb 2009, pp. 873–876.
- [52] OCG, "Sensor Model Language (SensorML) Implementation Specification," <http://www.opengeospatial.org/standards/sensorml>, Open Geospatial Consortium (OGC), 2007, Sensor Model Language (SensorML) Implementation Specification. [Online]. Available: <http://www.opengeospatial.org/standards/sensorml>
- [53] C. Jennings, Z. Shelby, and J. Arkko, "Media types for sensor markup language (SENML)," Working Draft, IETF Secretariat, Tech. Rep., Jan. 2013. [Online]. Available: <http://tools.ietf.org/html/draft-jennings-senml-10>

PAPER D

Plant descriptions for engineering tool interoperability

Authors:

Oscar Carlsson, Daniel Vera, Jerker Delsing and Bilal Ahmad

Reformatted version of paper originally published in:

Conference paper, IEEE INDIN 2016, Poitiers, 2016.

© 2016 IEEE. Reprinted, with permissions, from Daniel Vera, Jerker Delsing and Bilal Ahmad, *Plant descriptions for engineering tool interoperability*, IEEE INDIN, 2016.

Plant descriptions for engineering tool interoperability

Oscar Carlsson, Daniel Vera, Jerker Delsing and Bilal Ahmad

Abstract

The emergence and deployment of connected devices in many domains of application (e.g. industrial production, buildings and facilities, urban environment, etc.) have resulted in the need to achieve integration of multiple and more complex systems. This new environment is stressing the intrinsic limits imposed by monolithic standards, data models and integration methods that focus on specific domains of application, types of systems, or specific aspects of a system.

This paper describes the Plant Description Service developed as part of the Arrowhead Interoperability framework (EU ECSEL funded project). The manuscript contains a description of the abstract system descriptive model based on which the Plant Description service was implemented, and describes how the service can be used to achieve integration of several industry standards and data models. One use case and one case study is provided that illustrates how the service was practically implemented to support engineering scenarios in the domain of industrial production. The paper concludes with a critical review of the approach and suggestion for future work and developments.

1 Introduction

Large modern projects, such as construction of factories, power plants, airports, or railroad tunnels, incorporates many engineering disciplines and contains a large number of connected devices. In such projects the engineering quickly becomes a complex operation with the need to exchange data between different tools and data sources used by engineers from different disciplines. [1]

In many situations where safety and performance are critical, it is vital that components can be tracked throughout the complete life-cycle, including production, operation, maintenance, re-use and other possible scenarios. Pátkai et al. [2] illustrate this in a case study from the aerospace industry. Using unique identifiers both for the components and for the functional and locational sections that components are associated with lowers the risk of confusion as systems or personnel from different areas or disciplines are required to cooperate and exchange information. With end-to-end engineering, horizontal integration and vertical integration seen as overarching aspects of the German initiative Industrie 4.0 [3, 4] and with Innovative Engineering being seen as one of the main aspects of digital technology, as perceived by leaders in economy and society according to ITEA-ARTEMIS [5], there is support for more powerful and better integrated engineering tools.

In the status report on Reference Architecture Model Industrie 4.0 (RAMI4.0) [4] there is an emphasis on the combination of life cycle aspects, IT representation layers and the traditional automation hierarchies.

For the improvement of engineering tools in the field of industrial automation there is already considerable effort. For the process industry Braaksma et al. [6] have reviewed a large group of standards for asset information, with a focus on collaboration between engineers of different disciplines and information hand-over between different life-cycle phases. For the area of software engineering in industrial automation Vyatkin [7] presents an overview of the current solutions and concepts, where the role of standards such as IEC 61499 [8] and IEC 61850 [9] are highlighted, and the use of model-driven software engineering in automation is presented as one of the compelling paths for further development in the area. One example of model-driven software engineering that uses structured standards would be the method presented by Pang et al. [10] for how the Piping and Instrumentation Diagrams (P&ID's) following the standard IEC 62424 [11] can be translated into IEC 61499 Function Blocks (FB's) commonly used in programming automation systems for the process industry.

Himmler et al. [12] have developed a function-based engineering framework, illustrating how a structured and standardized description of a large system can improve the interdisciplinary collaboration throughout the engineering of a plant. As this framework uses a strict functional hierarchy as its basic structure for description of the plant it will present a view that is familiar to many process and automation engineers but if it is the only structure that can provide a good overview of the plant it may become cumbersome to other disciplines such as technicians, maintenance engineers and others that are more concerned with physical hardware. It is to a large degree due to these differences that there may be several different hierarchies present in an existing plant. In the worst case scenario there may be several different names for the same object, resulting in confusion and difficulties in communication between different disciplines.

Most approaches to engineering tool interoperability and standardization of engineering data exchange assume that all data to be exchanged will be harmonized around one standard identifying the assets that the data relates to. However, considering the large number of standards that are already in use, often enforced by engineering tools, the likelihood of one standard dominating all engineering data that concerns a large, modern, automated facility is not very high for the near future. To alleviate the situation with several standards at the same site there are already some initiatives on specific synchronization between pairs of complementing standards such as collaboration between Automation ML (IEC 62714) and OPC-UA (IEC 62541) [13] as one example and collaboration between ISO 15926 and Mimosa [14] as another. However, most standards for plant topology propose one primary aspect around which the main hierarchy is formed. In some cases the standards support additional, supplementary hierarchies as well but the main one is usually mandatory and used to order the data in a tree structure for exchange between systems. This is why we propose a simplified solution for describing the different hierarchies and topologies that may exist within the same plant or system of systems, sometimes defined according to different standards or procedures. The Plant

description aims to provide a basic common data structure which can be used to refer to different objects in a large system or system of systems and the relations between the objects.

The Plant description services are intended to give a basic common understanding of the layout of the plant or site, providing possibilities for actors with different interests and viewpoints to access their view of the same data-set. In the case of device replacement this is useful for the technician replacing the device to assign which position the new device is in, e.g. which old device it is meant to replace. To provide the option to view the same set of objects in different ways, arranging them in different hierarchies or networks depending on the desired viewpoint the basic data structure is proposed to be based around nodes and links. An example of how a traditional hierarchy could be represented can be seen in Figure 1. Once the objects are identified other tools, services or systems are intended to provide detailed information, based on the object identity provided by the Plant description. Other systems anticipated to provide such information are the systems managing configuration, services for orchestration, systems for meta-data and specialized engineering tools, but there are other possible services as well.

The main benefit of the proposed solution is intended to be a lower risk for misunderstandings between different organizations and disciplines without the need to force all involved parties to implement one standard that works for all purposes. Possible additional benefits include a lower threshold for utilizing engineering and design data from different sources that may be organized according to different standards.

2 Existing standards and related work

A number of interesting standards for exchange of engineering data have been identified and discussed. However, there is no clear solution for the purpose of providing a basis for interaction between engineering tools of the wide spectrum of domains and disciplines covered by the Arrowhead project, including but not limited to production facilities, building automation, infrastructure, electric vehicles and energy systems.

The Reference Architecture Model Industrie 4.0 (RAMI4.0) status report [4], a product of the German initiative Industrie 4.0, is centered around the standards IEC 62890 for structuring the Life Cycle and Value Stream, combined with the two standards IEC 62264 (ISA-95) and IEC 61512 (ISA-88) for structuring the hierarchical levels. At a more detailed level the report suggests a number of standards for different aspects. For implementation of the Communication layer the report suggests OPC-UA, for the Information layer IEC 61360 (ISO 13584-42), eCl@ss, Electronic Device Description (EDD) and Field Device Tool (FDT) are suggested. Field Device Integration (FDI) is suggested as integration technology and for end-to-end engineering the report suggests ProStep iViP, eCl@ss and AutomationML (which uses a topology based on IEC 62424).

OPC-UA (IEC 62541) [15] is the data exchange standard for platform and vendor independent communication across vertical and horizontal layers within industry in a client-server environment. OPC UA defines generic services and in doing so follows the design paradigm of service-oriented architecture (SOA). In contrast to classic Web

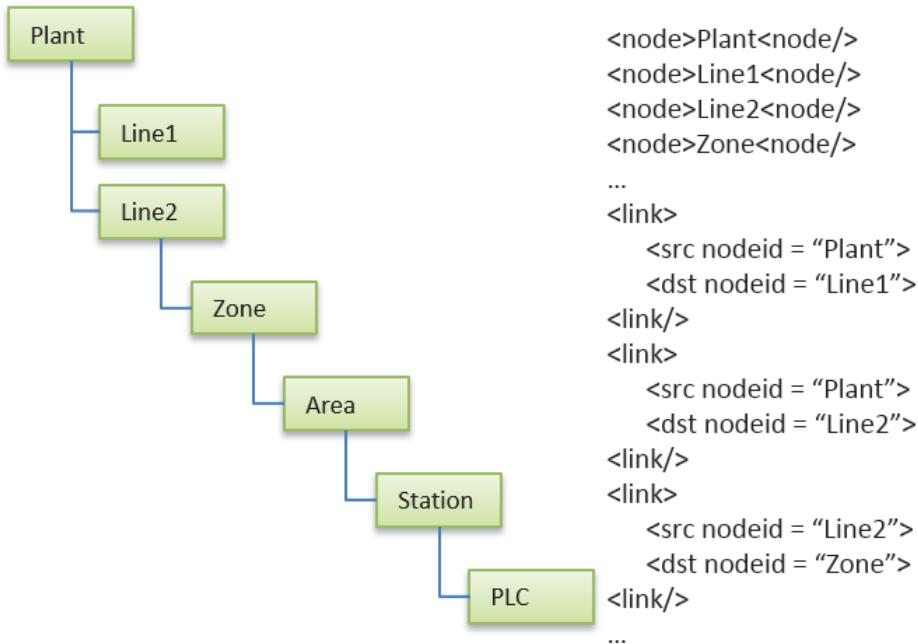


Figure 1: Hierarchy described by a nodes-and-links data structure

services, a number of generic services are already defined and standardized and thus WSDL is not required. Services are organized into logical groupings called service sets. Service requests and response are communicated through message exchange (either using binary protocol on TCP/IP or as a web service) between client and server.

ISO Technical Committee 184 for Automation systems and integration (ISO/TC184) has issued a large number of standards in the field of industrial automation systems [16], including: (in cooperation with IEC) IEC 62264, also known as ISA-95 [17]; ISO 15926 [18] for representation and exchange of life-cycle data of industrial process plants; ISO 10303 for Product data representation and exchange; ISO 15531 for Industrial manufacturing management data. The same technical committee has also released a draft for ISO 18828 that will standardize Manufacturing process and management information.

The ISO 15926 standard does allow for multiple disciplines and provides strong support for management of types, classes and instances of objects throughout the complete life-cycle of a process plant. However the ISO 15926 standard is very extensive, as Holm et al. [19] illustrate in the comparison between IEC 62424 and ISO 15926, through the representation of a belt conveyor according to both structures.

The IEC 62264 (ISA-95) is a commonly used standard that defines the hierarchical structure of interaction between an industrial control system and enterprise systems, specifically the functional data flow and object models. However the standard does not

in great detail specify the engineering data of the control systems and does not go into the interaction between engineering data of the control system and engineering data of electrical, mechanical or other systems that the control system by necessity are related to.

The ISO/IEC/IEEE standard 15288 [20] specifies a number of concepts that can be useful in the engineering of a large system, while not going into the details of each domain. This standard contains many useful concepts but is not yet widely adopted by the industry and still requires further details for fruitful interoperability between engineering tools of different disciplines.

The standard IEC 81346 describing Industrial systems structuring principles is common for identifying systems and objects in electrical installations within European industries, and is to some extent used within automation systems in such facilities for naming and structuring objects connected to the automation systems. In a similar manner IEC 61850 is used in electrical substation automation systems. The IEC 61850 data model has been mapped to standardized protocol DNP3 (Distributed Network Protocol) [21] for interaction with other automation systems and some interoperability with IEC 61499 has been shown by Yang et al. [22].

The work by Chen and Lin [23] on a Digital Equipment Identifier (DEI) system, which intends to uniquely identify manufacturing equipment and organize data retrieved from vendor Web sites or databases, could be seen as a potential further standard that can be used to describe systems at an automated production site.

As can be seen there are a great number of different standards that in some way concern the modularity of a large production system, by dividing it into objects, functions, locations or systems. To a large extent the modules are likely to be comparable, representing the same physical component, but there may be cases where certain components are neglected as modules for one discipline while very important for another.

3 Engineering tool interoperability

As with the interoperability of the kind of systems that are the main target of Arrowhead as a whole, the interoperability of engineering tools is possible today – as long as all partners follow the same standard or use the same suite of tools from one system provider [24]. However, as the Arrowhead project targets widely different domains [25], including different kinds of production facilities, building automation, infrastructure, mobile system such as electric vehicles and energy systems; and the engineering tools of those domains have to cover different aspects and life-cycle phases, there are many standards that could be used depending on the domains and life-cycle phase. A first stage of enabling broader interoperability between engineering tools will be to identify a basic form of interaction using services between different tools.

As many domains already follow standards relating to one or more of the aspects that this task aims to address it seems unreasonable to make all of them follow one standard. Instead some inspiration can be taken from the discussions that have already been had within Arrowhead regarding communication protocol translation [26], where the efforts

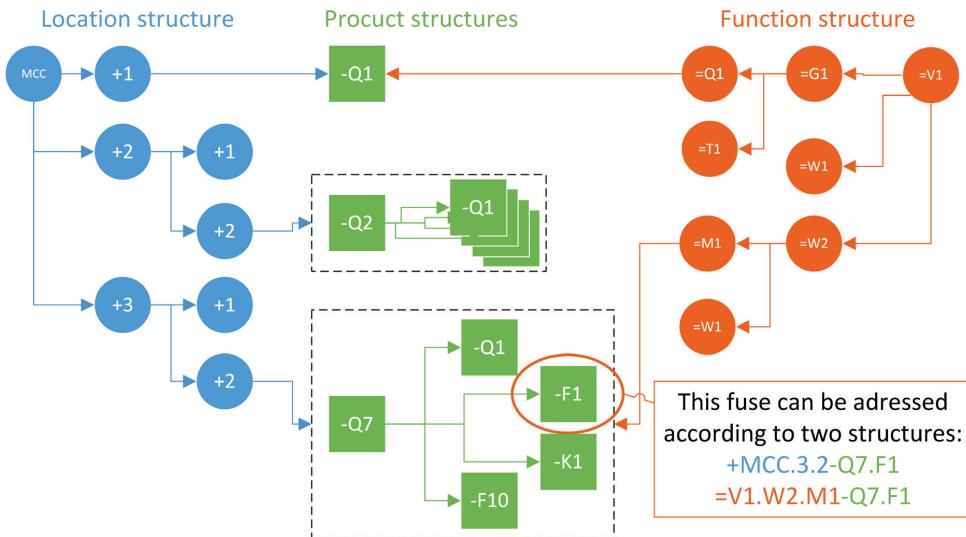


Figure 2: In IEC81346 the three aspects Function, Location and Product are commonly used to describe different viewpoints of industrial systems

have been concentrated on a solution using one protocol as an intermediary layer and focus on translating to and from this one protocol rather than direct translation between all protocols. Many of the standards concern the parametrization of the object data, which is important for compatibility between tools within one domain, but as the target here is interoperability between tools from different domains it may be sufficient to focus on a few key parameters for each object and the different relations the objects have between themselves.

3.1 As a service in the Arrowhead Framework

The purpose of the Plant Description service is to provide a basic common understanding of the layout of a plant or site, providing possibilities for actors with different interests and viewpoints to access their view of the same data set provided by other sources. A source of inspiration for the design was the standard ISO/IEC 81346 which specifies that for studying objects and their relations it may be useful to look at them from different viewpoints, highlighting different aspects of the objects and relations. This standard is focused on the three aspects: function, product and location, although the design is intended to be capable to address other viewpoints as well.

In order to be able to present all of the different types of objects and relations that are expected to be present in a future Internet of Things (IoT) or Industrial Internet of Things (IIoT) network in a useful engineering tool or set of tools the concept of displaying different aspects of the same objects appears to be a useful solution. Figure 2 illustrates

NodeId: ab:12:cd::01 - IEC81346-Function: =V1.G1	LinkId: cd:12::ff:78 Source: Nodeld: ab:12:cd::01 Target: Nodeld: ab:12:cd::02 LinkTypes: IEC81346-Function
NodeId: ab:12:cd::02 - IEC81346-Function: =V1.G1.Q1	LinkId: cd:12::ff:79 Source: Nodeld: ab:12:cd::02 Target: Nodeld: ab:12:cd::03 LinkTypes: IEC81346-Function
NodeId: ab:12:cd::03 - IEC81346-Function: =V1.G1.Q1-Q1 - IEC81346-Location: +MCC.1-Q1	LinkId: cd:12::ff:7a Source: Nodeld: ab:12:cd::04 Target: Nodeld: ab:12:cd::03 LinkTypes: IEC81346-Location
NodeId: ab:12:cd::04 - IEC81346-Location: +MCC.1	

Figure 3: An example of how the object -Q1 in Figure 2 and three connected nodes could be described by a data structure based on nodes and links.

how an object documented according to IEC 81346 can be part of a product, visualised in green, addressed either based on its location, visualised in blue, or the function it performs in the facility, visualised in orange.

3.2 Nodes and links

In an effort to limit the amount of data, the proposed solution will only store a few parameter for each object within the Plant description, and a few parameters for each link. Instead, the solution will rely on other data sources, such as existing databases and engineering tools, to provide more specific data.

Therefore the data stored about each object is limited to (1) a unique identifier used within the Plant description and (2) a list of data pairs containing one pair for each standard or other system in which the object is identified and its identifier within that standard and system. Each of these objects is considered a node. Similarly each relation between two objects will be stored as a link, containing four data items: (1) a unique identifier used within the Plant description system, (2) the identifier of the node that is the source of the link, (3) the identifier of the node that is the target of the link and (4) a list of standards and systems in which these two nodes are linked to each other.

Figure 3 illustrates how the nodes =G1, =Q1, -Q1 and +1, and their relations, as found in the top row of Figure 2, can be described by a set of nodes and links.

4 Plant description use cases

The Plant description is intended to have multiple usage areas. This section describes one envisioned use case and one case study where a prototype implementation provided some specific required functionality.

4.1 Design and commissioning

In a common scenario a group of engineers from different disciplines collaborate to design an automation System of Systems, based on commercially-of-the-shelf (COTS) available automation components, using different engineering tools and standards as appropriate to their respective disciplines. Main requirement is that all engineering tools use a modular approach, identifying components as modules, objects, subsystems or similar, that can be combined into a larger automation system.

The engineers will, throughout the whole process, use the Plant description as a reference tool for objects and systems that have been identified and how they are related to each other. At the early stages the Plant description can help engineers from different disciplines to coordinate their work, even though the design, names of objects, and responsibilities of subsystems may not be fully decided yet.

As engineers from difference disciplines start to populate the design with specific objects, the data can be synchronized between design tools using the Plant description. Throughout the design phase, all of the engineering data is still stored and maintained in the formats preferred by the engineering tools in their respective databases. The data in the Plant description should be limited to what objects exist in the separate databases, how they are identified and what their relations are.

As the systems become ready for commissioning the Plant description can allow technicians to navigate the design using the structure that best fits their knowledge and requirements, while still having the possibility of accessing engineering data from the respective disciplines. Once the systems are commissioned there can be made a direct link between the actual hardware and software on the device and the data from the design and engineering process.

The use of the Plant description should lower the risk of misunderstandings and help identify cases, primarily during design and engineering, where different disciplines use different names for the same object.

4.2 Automotive industry

In an effort to implement the concepts and vision of Industrie 4.0, Ford Motor Company, UK is currently undergoing a development phase whose purpose is to achieve integration of the increasing number of interconnected devices deployed in power train assembly plants. In addition to PLC (Programmable Logic Controllers) devices used for control automation, several other types of connected devices are being deployed on the production system itself or linked to various other assets in the shop floor; e.g. RFID tags/antenna systems fitted on material transport and storage (i.e. pallets, racks) for tracking material

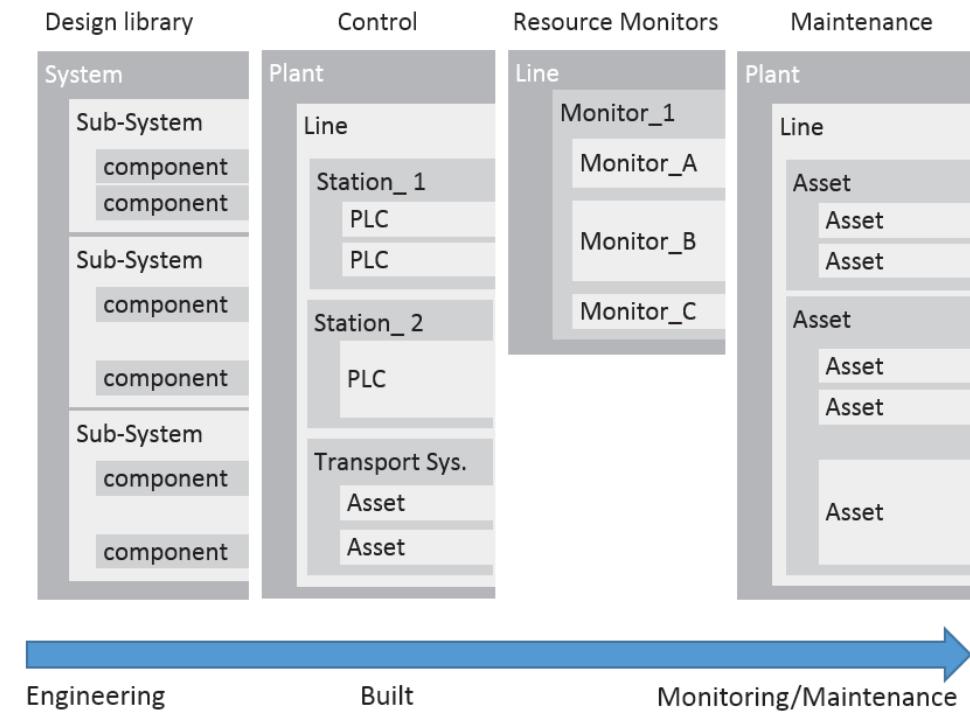


Figure 4: Four different system hierarchies as used in the automotive industry

flow through the shop floor, smart tooling and wearable sensors and trackers, deployment of Resource and Energy Monitors for collecting contextual data (energy, temperature, vibration). Most devices used are commercial-off-the-shelf products whose deployment and operation rely on specific and often proprietary data structure, connectivity protocols and DBMS (Data Base and Management Systems) back ends. Therefore the design and life-cycle management of future production systems has to focus on the implementation of engineering methods and tools that allow to deal with the increasingly complex nature of CPS.

Figure 4 is a simplified illustration of various data sets describing a same production line from various perspectives which are specific to either engineering domains (e.g. layout, mechanical, control engineering, etc.) or specific to various phases of its life-cycle (e.g. design, commissioning, operation/maintenance). Each views use specific data models (i.e. data types, formats and data structure). For instance, design libraries (PLM databases) contain product, process and resource (PPR) data used to support digital engineering. Various aspects of the system are typically described at different levels of granularity; For instance a single control PLC may control two or more processing areas

as defined in the design library (see Figure 4). Other devices networks such as resource monitoring devices may only be deployed on a sub set of the entire production line so that some aspects of the system are only partially defined compared to others. Heterogeneity at the physical level is reflected in the data models used to store and structure information/data generated by or describing the system. For instance the data structures in both design libraries and maintenance databases (MAXIMO [27] as used by Ford globally) are not consistent as they are targeted at different activities and implemented for different purposes. In addition, the identification of devices which are physically or functionally linked may also be inconsistent: For instance, brass plates and 2D tags are used to uniquely identify physical assets in the shop floor while PLCs and other connected devices typically use IP addresses as unique identifiers. The same assets or devices also exist as a uniquely identified entry in Data Bases.

In this context, the semi-generic descriptive model provided by the Plant Description service was used to a) capture the structure of various sub systems and network of devices that compose complete production lines and facilities and b) map different assets and devices' ID using the Nodes and Links element as described in section 3.2. This mapping information is essential in enabling navigation and correlation between various data sets and therefore in enhancing cross engineering domains communication and achieve better integration of engineering processes.

The Plant Description model and ID mapping information were used to support the implementation of a so-called Fault Tracker application developed by WMG in the University of Warwick, as described by Harrison et al. [28]. The Fault Tracker mobile application is used in the shop floor in order to support preventative and fault-fixing maintenance of production systems. Figure 5 illustrates how the Fault Tracker is used to aggregate data from different data bases into rich, context sensitive user views supporting various phases of maintenance procedures; a) Fault information and/or maintenance work orders are retrieved from the maintenance database (MAXIMO in this case); based on the maintenance database ID identifying the faulty asset, the b) location of the faulty asset is highlighted in the plant and shop floor layout 2D or 3D CAD data retrieved from the layout engineering database. This information is used to provide location and guidance to the faulty system; c) Views of the faulty component and contextual information are provided (e.g. electrical diagrams, vendor manuals, 3D CAD of faulty component, safety procedures, etc.) in order to assist diagnostic; c) The user can then upload updated maintenance information and/or fault fixing information back to the maintenance database and conduct additional actions such as ordering parts or update the maintenance schedule.

The Fault Tracker case study aims at demonstrating the value of the descriptive model (i.e. Plant Description) overarching several aspects of a same system and of the associated services (e.g. ID mapping) that can be used to identify and retrieve various sets of data and information in order to better support a specific use case (i.e. maintenance of automation systems). The present implementation allows aggregation of maintenance, 2D layout, 3D engineering and control data and presentation into a rich and task specific mobile application user interface. Future implementation will focus on integration of

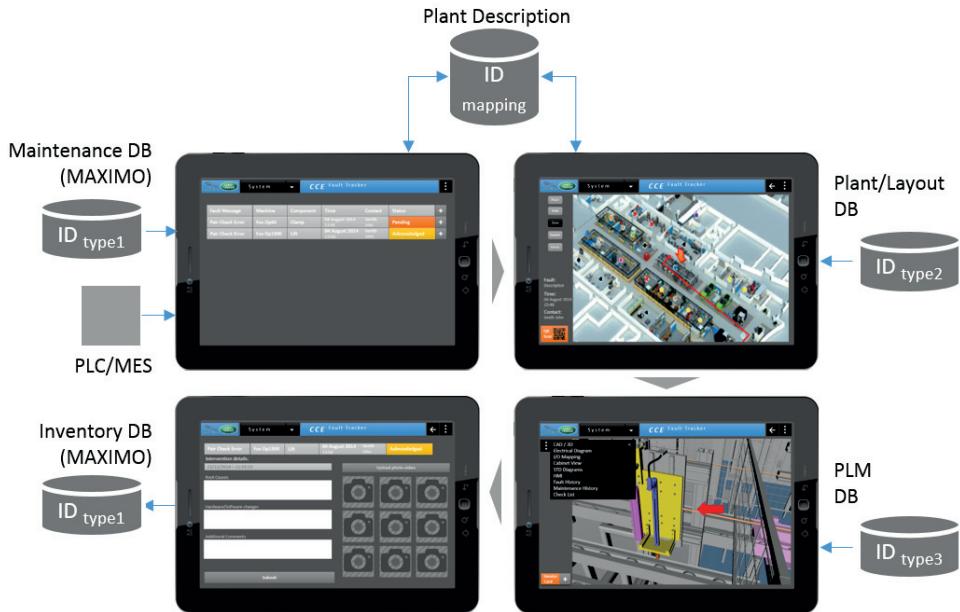


Figure 5: Fault Tracker maintenance application screens and data sources

additional data sources e.g. energy monitors, location tracking. Prototype implementation and testing were conducted using the full scale Automation System Workbench demonstration system at WMG, University of Warwick which allowed Ford Motor engineers to identify potential benefits: reduced time to locate and attend asset, reduced time to identify fault, permanent access to up-to date and context/task dependant information, monitoring and traceability of maintenance operations, better management of maintenance personal, paperless work orders management.

5 Conclusion

The initial purpose of the solution presented in this paper was to provide a simple service interface to navigate different aspects of the standard IEC 81346, most notably to be able to switch between the function aspect and the location aspect, without having to provide all of the detailed data included in each object.

The solution presented in this paper constitutes an alternative for the exchange of engineering data which does not force all systems to use the same standard or require full compatibility between all relevant standards. The solution provides a basic structure on top of which further compatibility between engineering standards can be developed. Compared to traditional solutions, this solution is more flexible towards the designers of engineering tools but at the cost of adding the additional Plant description data set that

needs to be defined and managed. Ideally, the design should be possible to extract from existing engineering data but there is likely some manual identification or matchmaking that needs to be performed as structures from different tools or standards are coordinated.

One aspect of future work would be the identification of objects present in more than one standard to make sure that they are not duplicated in the Plant description. While it is fairly simple to automate the conversion of XML-based topological trees that are used in many standards to nodes and links there may still be the need of considerable engineering effort in synchronizing the nodes created from different standards and to identify possible duplicates.

The future work should also include the design of a full engineering procedure, using the Plant description for integration and coordination. Such a procedure is envisioned to include deployment and programming or configuration of devices into facility, possibly also including management of security features. As a more mature implementation of the Plant description is also part of the future work, this will enable more case studies and more detailed evaluation of the benefits and drawbacks of the approach.

Acknowledgment

The authors would like to thank all partners of the Arrowhead project for the fruitful discussions and the ARTEMIS JU and ECSEL JU for funding, within project Arrowhead, grant nr. 332987.

References

- [1] R. Yang, *Process Plant Lifecycle Information Management*. AuthorHouse, 2009. [Online]. Available: <https://books.google.se/books?id=zmgzdjf1GR0C>
- [2] B. Pátkai, L. Theodorou, D. McFarlane, and K. Schmidt, “Requirements for rfid-based sensor integration in landing gear monitoring—a case study,” *Power*, vol. 5, p. 4, 2007.
- [3] J. H. Henning Kagermann, Wolfgang Wahlster, “Recommendations for implementing the strategic initiative INDUSTRIE 4.0,” acatech - National Academy of Science and Engineering, Tech. Rep., 2013.
- [4] P. Adolphs, H. Bedenbender, D. Dirzus, M. Ehlich, U. Epple, M. Hankel, R. Heidel, M. Hoffmeister, H. Huhle, B. Kärcher, H. Koziolek, R. Pichler, S. Pollmeier, A. Walter, B. Waser, and M. Wollschlaeger, “Status Report - Reference Architecture Model Industrie 4.0 (RAMI4.0),” VDI - Verein Deutscher Ingenieure e.V. and ZVEI - German Electrical and Electronic Manufacturers’ Association, Tech. Rep., July 2015. [Online]. Available: <http://www.zvei.org/Downloads/Automation/5305PublikationGMAStatusReportZVEIReferenceArchitectureModel.pdf>

- [5] ITEA and ARTEMIS-IA, "ITEA ARTEMIS-IA High-Level Vision 2030 - Opportunities for Europe," ITEA and ARTEMIS-IA, Tech. Rep., 2013.
- [6] A. J. Braaksma, W. W. Klingenberg, and P. P. van Exel, "A review of the use of asset information standards for collaboration in the process industry," *Computers in Industry*, vol. 62, no. 3, pp. 337 – 350, 2011. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0166361510001387>
- [7] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 3, pp. 1234–1249, Aug 2013.
- [8] *International Standard IEC61499, Function Blocks, Part 1 - Part 4*, IEC Std.
- [9] *IEC 61850 - Communication Networks and Systems in Substations*, IEC Std.
- [10] C. Pang, V. Vyatkin, and W. Dai, "IEC 61499 based model-driven process control engineering," in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, Sept 2014, pp. 1–8.
- [11] *IEC 62424: Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools*, IEC Std.
- [12] F. Himmller, M. Gepp, J. Vollmar, T. Jager, and M. Amberg, "Function-based engineering framework for the standardization of industrial plants," in *Industrial Electronics Society, IECON 2014 - 40th Annual Conference of the IEEE*, Oct 2014, pp. 4909–4915.
- [13] M. Schleipen. (2014) Open standards for Industry 4.0 – Tools and offer around AutomationML and OPC UA. [Online]. Available: http://www.iosb.fraunhofer.de/servlet/is/46944/AutomationML_en.pdf
- [14] A. T. Johnston, "OpenO&M and ISO 15926 Collaborative Deployment," October 2009. [Online]. Available: <http://www.mimosa.org/presentations/openom-and-iso-15926-collaborative-deployment>
- [15] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer, 2009.
- [16] Y. Fukuda, "Standardization for manufacturing systems in iso," in *ICCAS-SICE, 2009*, Aug 2009, pp. 955–958.
- [17] *ISA95, Enterprise-Control System Integration*, ISA Std.
- [18] *ISO 15926 Industrial automation systems and integration-Integration of life-cycle data for process plants including oil and gas production facilities*, ISO Std.

- [19] T. Holm, L. Christiansen, M. Goring, T. Jager, and A. Fay, "Iso 15926 vs. iec 62424 - comparison of plant structure modeling concepts," in *Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on*, Sept 2012, pp. 1–8.
- [20] *ISO/IEC/IEEE International Standard - Systems and software engineering System life cycle processes*, Std., Jan 2008.
- [21] "Ieee approved draft standard for exchanging information between networks implementing iec 61850 and ieee std 1815(tm) (distributed network protocol - dnp3)," *IEEE P1815.1/D8.00, September 2015*, pp. 1–356, Jan 2015.
- [22] C.-W. Yang, J. Xu, and V. Vyatkin, "Towards implementation of iec 61850 goose messaging in event-driven iec 61499 environment," in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, Sept 2014, pp. 1–4.
- [23] T. Chen and Y.-C. Lin, "A digital equipment identifier system," *Journal of Intelligent Manufacturing*, pp. 1–11, 2015. [Online]. Available: <http://dx.doi.org/10.1007/s10845-015-1071-3>
- [24] R. Harrison, D. Vera, and B. Ahmad, "Engineering methods and tools for cyber-physical automation systems," 2016, Proceedings of the IEEE, Accepted October 2015 (In Press).
- [25] F. Blomstedt, L. Ferreira, M. Klisics, C. Chrysoulas, I. Martinez de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, "The arrowhead approach for soa application development and documentation," in *Industrial Electronics Society, IECON 2014 - 40th Annual Conference of the IEEE*, Oct 2014, pp. 2631–2637.
- [26] H. Derhamy, J. Eliasson, J. Delsing, P. Pereira, and P. Varga, "Translation error handling for multi-protocol soa systems," in *Emerging Technologies Factory Automation (ETFA), 2015 IEEE 20th Conference on*, Sept 2015, pp. 1–8.
- [27] IBM. IBM Maximo Asset Management. [Online]. Available: <http://www.ibm.com/software/products/en/maximoassetmanagement>
- [28] R. Harrison, D. Vera, and B. Ahmad, "Engineering methods and tools for cyber-physical automation systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 973–985, May 2016.

PAPER E

Organizing IoT Systems-of-Systems from Standardized Engineering Data

Authors:

Oscar Carlsson, Csaba Hegedűs, Jerker Delsing and Pal Varga

Reformatted version of paper originally published in:

Conference paper, IEEE IECON 2016, Florence, 2016.

© 2016 IEEE. Reprinted, with permissions, from Csaba Hegedűs, Jerker Delsing and Pal Varga, *Organizing IoT Systems-of-Systems from Standardized Engineering Data*, IEEE IECON, 2016.

Organizing IoT Systems-of-Systems from Standardized Engineering Data

Oscar Carlsson, Csaba Hegedűs, Jerker Delsing and Pal Varga

Abstract

Tackling current challenges in production automation requires the involvement of new concepts like Internet of Things, System-of-Systems and local automation clouds. The objective of this paper is to address the actual process of defining a cloud based automation system. More specifically the design, engineering, operation and maintenance of an automation system must be captured and managed between all stakeholders involved. This is critical to create the expected benefits from the local automation cloud approach.

This paper addresses the capability of capturing plant designs and coordinating information exchange based on the captured architecture. For this purpose an architectural component – Plant Description – is proposed to be used in the Arrowhead Framework, based on already existing plant automation standards. An overview of methodologies on how it can interact with the Arrowhead Framework’s Orchestration process describes the usefulness in managing large-scale automation systems. A qualitative evaluation for one of the proposed approaches is also described in a water control use case that can be found both in process and building automation.

1 Introduction

Many of the current societal challenges revolve around efforts for sustainability, flexibility, efficiency and competitiveness. This trend, among others, is imposing new requirements on the technologies used to support production. Here a number of gaps have to be addressed regarding e.g. technologies, related business models and operational management.

Despite all the new operational and organisational ideas, the automation fundamentals captured by today’s state-of-the-art automation technologies have to be maintained. Thus the next generation of automation and digitalisation technology has to meet a large set of requirements and involving a wider scope of actors and stakeholders.

In 2011, the concept of Industry 4.0 [1] was born in Germany. This concept builds upon the last generation of industrial monitoring and control systems, but enables an even finer level of interaction between shop-floor devices and high-level enterprise systems. In Industry 4.0, Internet of Things (IoT), Cyber-physical Systems (CPS) and other concepts are to be utilized in order to break down the classical strict hierarchical approach of ISA-95 [2], replacing it with a more flexible approach without barriers and closed systems.

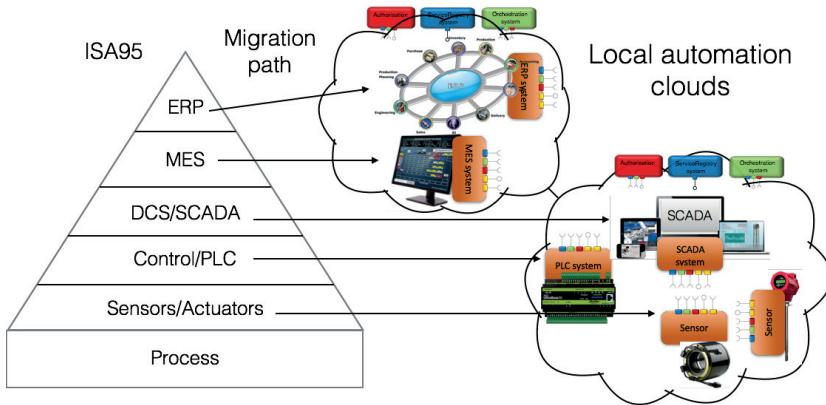


Figure 1: Transferring the ISA-95 automation pyramid architecture to a local automation cloud architecture.

For the transfer of the ISA-95 architecture to a cloud based approach much important work is already published, including: system architectures [3, 4]; suitable technologies [5]; real time considerations [6, 7]; migration from legacy systems [8, 9] and engineering for cloud based automation [10, 11]. This cross-compliance is depicted on Fig. 1 based on approaches of the SOCRADES [12] and IMC-AESOP [13] projects. Moreover, a parallel discussion is also present for the MES and ERP levels. Some important works on cloud approaches are [3] and [14].

The concept of local automation clouds have recently been introduced via the Arrowhead project through the newly released Arrowhead Framework [15] and the book edited by Delsing [16]. Automation clouds have a number of important properties to consider. One group of its properties is based on how a “production plant” is described and viewed. Such a plant descriptor tool should be capable of addressing a number of application requirements and therefore should reflect the engineering, deployment, operation and maintenance features of the “plant”. The following list enumerates a couple of usage scenarios:

1. Plant design and System of Systems interaction design at design time
2. Management of run-time changes made in the plant, i.e. replacement of devices or rearrangement of the System of Systems interaction
3. Supporting automated deployment procedures, plant commissioning, plant restart and plant updates
4. Run-time creation of System-of-Systems orchestration rules based on physical processes
5. Extraction and monitoring of current plant status

6. Supporting run-time error and fault handling processes, possibly with alternative orchestration configurations
7. Optimizing production and physical processes

The objective of this paper is to primarily address the forth and fifth items of this list. Thus the detailed objectives are to explain how to capture “plant” changes at run-time and distil updated System-of-Systems (SoS) orchestration rules based on them. It is also of interest to recognize “plant” operation errors and faults requiring reconfiguration of the SoS within the Arrowhead framework.

To meet these, section 2 outlines the related work, followed by section 3, which presents the proposed approach and the integration within the Arrowhead framework. Section 4 describes a practical use case, and then Section 5 discusses the conclusions and maps out the future work.

2 Related work

2.1 Process Engineering State of the Art

There are already several standards and groups of standards that allow the integration of engineering data from different sources. Although they use different approaches, most of them appear to aim for full integration of all data into one large database. Yang describes many of the challenges and strategies that has fostered this approach [17].

One example is the group of standards put forward by the ISO Technical Committee 184, Subcommittee 4 (ISO/TC 184/SC 4)¹. They cover many different aspects of industrial information and data, and also include the ISO 18876 that presents a methodology specifically for integrating industrial data based on different data models into one extended data model.

Another such group of standards is based around IEC 62714 (AutomationML) [18, 19]². AutomationML uses a tree-structure that is used to organize all objects to describe the plant topology. This gives the defined hierarchy higher priority than any other object relations that can be represented by using separate link-objects called CAEX-InternalLinks [20]. Lüder et al. [21] have shown how additional information (not specified in the standard) can be appended to an existing AutomationML model, allowing all engineering data to be contained in one consistent data set.

The former group of standards appears to be more popular in the process industries and the American defense industry, while the latter appears to be more popular in manufacturing industries. Holm et al. [22] presents a comparison of the plant modeling concepts described in ISO 15926 and IEC 62424, respectively.

¹This group includes the ISO 8000, ISO 10303, ISO 13584, ISO 15926 and ISO 18828 standards.

²Such as IEC 62424 (CAEX), ISO/PAS 17506 (COLLADA) and PLCoopen.

2.2 The Arrowhead Framework

The Arrowhead framework uses a service-oriented approach to tackle interoperability and integrability issues within closed or separated automation environments called *local clouds*. It does so by facilitating and governing the service interactions between *Application Systems* using the *Arrowhead Core Systems*, see Figure 2 [16].

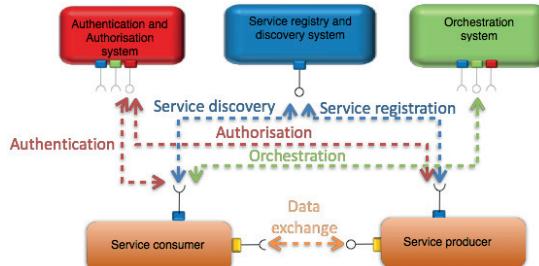


Figure 2: Arrowhead Core Systems and their use

A *service* in the Arrowhead Framework is what is used to exchange information from a providing System to a consuming System, e.g. a sensor readout or a command for an actuator. The Core Systems of the Arrowhead Framework help to establish and manage the service connections, but the service interaction is performed directly between the application systems. There are three mandatory Core Systems that are required to form a local cloud, as depicted in Figure 2 – however additional core systems can be used to support different aspects of a local cloud, the Plant Description System being one of them.

The Service Registry stores the service offerings of each registered system. As its name suggests, the Authorization System manages AA tasks. Meanwhile, the Orchestrator is responsible for instrumenting each System in the cloud: where to connect and what to consume. It instructs so by pointing towards specific Service Producers to consume specific Service(s) from (giving hence an *orchestration rule*). This orchestration service can, however, be provided through various internal methods (tailored to use cases), as discussed in section 3.2. The orchestration rule itself merely has to point to a Service Producer and declare the Service that is to be consumed at minimum. If required, further information can and should also be passed on (i.e. authorization information, service metadata or validity period).

The Plant Description

A “plant” in this context within the Arrowhead framework refers to a larger group of systems acting together, often defined by a geographical or organizational boundary. A typical plant would be a factory, a mine, or a water treatment plant, but it could as

well be a hospital, a windmill farm or an airport. For many purposes it is necessary to group systems together into hierarchies, and describe how the systems relate to each other. There are already numerous standards exist to describe these hierarchies, but since many disciplines are required to build a plant, there are often many different hierarchies from different standards within one plant.

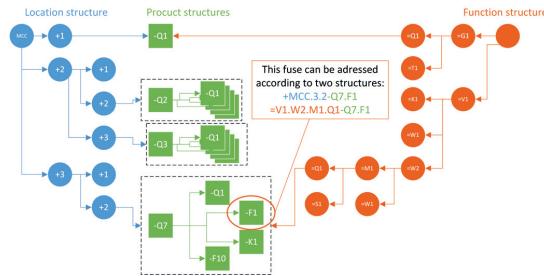


Figure 3: Example of how objects may be identified by more than one viewpoint, based on IEC 81346 [23]. The green objects in the middle are groups of electrical equipment that are both located together and perform a function together.

Figure 3 illustrates how a device or component – documented according to IEC 81346 [23] – may be found through traversing two different hierarchies, depending on the interest of the user. In this figure the green objects in the middle are groups of electrical equipment (switches, contactors and fuses) that are both placed at the same location, and perform a function together. The role of this group of objects – as per the function they provide – can be derived by traversing the red hierarchy on the right of the figure, while their physical location can be found by traversing the blue hierarchy on the left. Each colour on the figure can be represented as a separate viewpoint by the Plant Description, i.e. blue representing the location of the object or green representing the components making up a product.

The objective of the Plant Description therefore has been primarily to provide a basic common understanding of the layout of a “plant”, providing possibilities for actors with different interests and viewpoints to access their view of the same data set. Details of the design and prototype implementation of a Plant Description are described by Carlsson et al. [24]. For its use as a source of design data for System-of-Systems organization, its most important features are (i) the simplified description of object relations, (ii) topologies, and (iii) a unified data storage node structure.

3 Approach

The creation of orchestration rules from relations between automation objects – as set by design and engineering tools – should be automated. Defining such an automated

process would simplify deployment and commissioning procedures, as well as enable run-time creation of orchestration rules. If set-up correctly it could also allow for creation of alternative orchestrations, that could be used to mitigate certain run-time errors and automate fault handling. This section is intended to present how the Plant Description could be used to automate the creation of orchestration rules.

3.1 Plant Description

The Plant Description is used for identifying the objects and their relations while leaving the object details in the already established, standardized databases provided by existing engineering tools. The objects are stored as nodes, containing only the identifiers used in the respective engineering databases. The object relations are stored as separate links between nodes, with an attribute to each link denoting what kind of relationship the link represents.

Figure 4 shows a simplified illustration of how the Plant Description can be used to construct an orchestration rule. To the left, in blue, are the nodes and links in the Plant Description describing automation objects and their relations. One of these links, called the Functional service link, describes that the automation object at the top is to have a service interaction with the automation object at the bottom. In other words there is a Functional relationship between those nodes at the top and bottom – and it is a service interaction. It is important to note that the automation objects in the Plant Description are merely logical representations and not the actual hardware and software that performs the automation. To the right, in green, in figure 4 are the actual Systems (i.e. a software running on some hardware in the physical plant). In order to create an orchestration rule between the top System and the bottom System, it is enough to have knowledge of the links and nodes in the Plant Description and the knowledge of which System represents which node. If both of these pieces of data are available to an “Orchestrator”, such rules can be created automatically.

An extension of the Plant Description that is only conceptual so far, is to allow a certain degree of flexibility in the connections between objects. This concept states that the Plant Description should describe alternative service links, so that some but not all of the designed service links can appear as orchestration rules. This can be done in more than one way but it requires either an additional parameter for each link, or that there should be different link-types for the different kinds of service links. For flexible orchestrations they could be described in the Plant Description either as multiple link-objects – each link connecting one source node to one target node –, or as single links containing multiple sources and/or targets, where not all connections should be realised as orchestration rules.

3.2 Orchestration of systems

Orchestration in the Arrowhead context refers to a supportive task, namely providing instructions to Application Systems on their service consumption behaviour, as discussed in section 2.2. Since this is merely a black-box definition for the Orchestration service

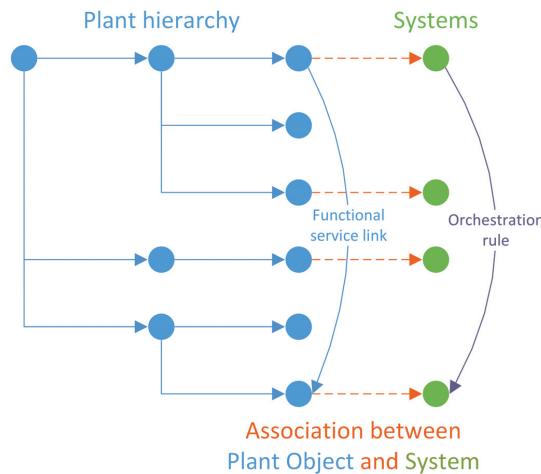


Figure 4: Illustration of how a relationship in the Plant Description can translate into an orchestration rule.

(describing its inputs, outputs, and interfaces), it provides room for various technological approaches on its internal logic and implementation. Therefore, a high-level design decision can be made on how to build the orchestration process that best matches service-consuming Systems to producing Systems. This degree of freedom can result in fundamentally different “Orchestrators” that fit the different needs of various use cases. Currently, there are three archetypes of Orchestrators in scope within Arrowhead. This section explores these and how they can be configured to operate in a “plant”.

Static rule-sets

In certain cases, static rule-sets are sufficient to orchestrate complex Systems-of-Systems since their compositions do not change over time. In this case, every system in the local cloud has a set of partners pre-defined and they cannot connect anywhere else. If this composition changes, however, it can only be carried out through human intervention in a process re-engineering procedure. The description of the System-of-Systems only changes during these reconstructions or tweaks. Here, the orchestration rule-set is also manually created.

For these cases, the Plant Description is rather a process engineering tool that supports these transitions. It describes how the SoS is laid out and realized with physical devices. This way, devices then can be mapped into systems in the Arrowhead aspect, and have their orchestration data re-entered in the orchestration store. Therefore, being user-friendly is a major requirement towards the plant description in this scenario.

As an example for this use-case, let us consider an actual production plant where Arrowhead services are physical processes and Arrowhead systems are e.g. actuators,

sensors or controllers. For such cases, certain sensors can only provide data for specific controllers and the same goes for actuators. These static restrictions on connections can be translated into orchestration rules. The task of the orchestrator is merely to support the (re-)establishment of these connections. Nevertheless, an Orchestrator using static rule-sets is still beneficial, since proper Plant Description tools can make it easy to reprogram a production line.

Dynamic orchestration

For those use-cases that are challenged by changing environments and system set-up, the Orchestrator system needs to become a dynamic matchmaker that brings together service consumers with providers at run-time. However, a such matchmaker should naturally not operate in a free-for-all manner. Even though Systems might be able to connect to every other System, such ad hoc connections might still not be desired (or e.g. authorized). Therefore, dynamic service discovery has to be augmented with admission control and limited by other factors – such as physical or (real-time) resource-related restrictions in the local cloud as depicted on Figure 5.

For these cases, the Plant Description functionality could provide general configuration information, as well as information of alternative rule-sets (e.g. describing alternative connections), and the preferences of their usage. Utilizing the functionality of different viewpoints in the Plant Description, the different Core Systems would be able to access the objects and relationships relevant to their functionality. This approach further eases the configuration for the Core Systems, as it reduces the risk of inconsistency in data used by different Core Systems. However, it does add further complexity to the input and management of Plant Description data.

Automated process engineering

In previous cases the orchestration process worked in a solicited manner: requested on pre-set criteria (e.g. events) or by service consumers. A third option is automating the process engineering, enabling unsolicited orchestration. The matchmaking of service consumers and providers is further optimized here by a response on the status of the SoS – based on measured operational variables. In this case the trigger of orchestration is not a direct request from any system; it rather comes from an inferred decision; based on the up-to-date knowledge about the SoS. The concept is very similar to the initial idea of Cognitive Networks, Software Defined Networking, or the Knowledge Plane approach [25].

The overall configuration of the System-of-System will be optimized for the global target of the given local cloud (e.g. setting a production target for production lines, or optimal energy consumption for a smart building). This sort of Orchestrator is then able to configure all systems to reach the higher goal: what services they need to consume, from where, or for how long. Such an Orchestrator engine would naturally have to utilize Plant Description related data and its operation must rely on the functional, (physical) architectural and process-oriented mappings of the SoS.

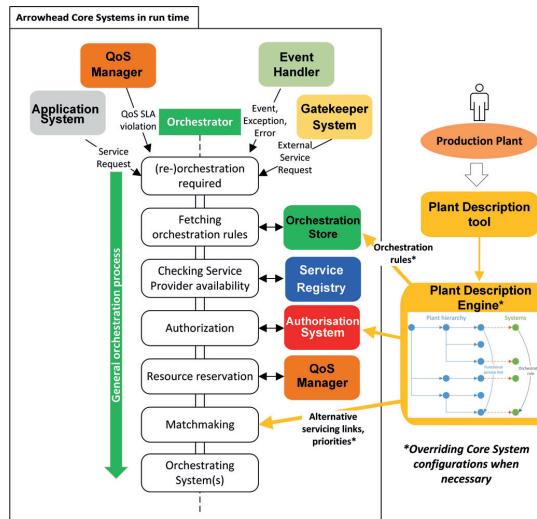


Figure 5: A Plant Description Engine providing configuration for the Core Systems

In such scenarios, the decisions and acts of such an engine is asynchronous to the general operations within a local cloud. It is only invoked to change the general flow in order to enforce the existing operational targets on the SoS (or push new ones). This means that the Plant Description in this case will be used to store required, desired or recommended connections between design objects.

4 Use case: Orchestration from Process & Instrumentation Diagram

Pang et al. [26] have shown how a system described by an IEC 62424 Process and Instrumentation Diagram (P&ID) can be formally transformed to IEC 61499 applications. Using part of the P&ID in their example to create a Plant Description, it can then be used to illustrate how orchestration rules might be generated for a System-of-System, based on service interactions.

Figure 6 shows the portion of the P&ID to be used in this example. As an implementation using the Arrowhead framework, this can be described as five systems – all belonging to the function “Water control loop” –, and between these systems there are four service interactions. In the Plant Description, these five systems would be considered Nodes, given a unique NodeId and an identifier within the CAEX-structure that would allow a user to access further information through there. Each link between these systems would be given a few parameters: (i) unique LinkId, identifiers for the link’s (ii) source and (iii) target nodes, and (iv) link type. An example of what the data in a Plant

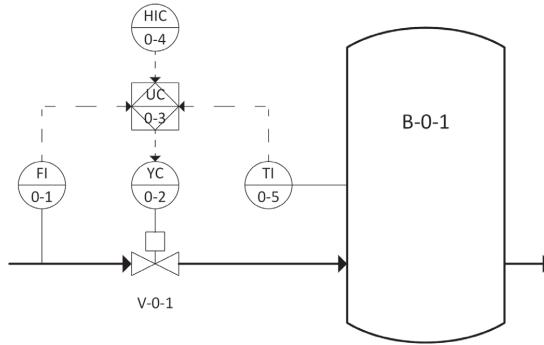


Figure 6: P&ID of a water control loop, including a Flowmeter (FI), a Thermometer (TI), a Control valve (YC), a Controller (UC) and a User interface (HIC); all used to control the flow through the Valve V-0-1 into the Tank B-0-1.

description might look like is given in the left part of Figure 7.

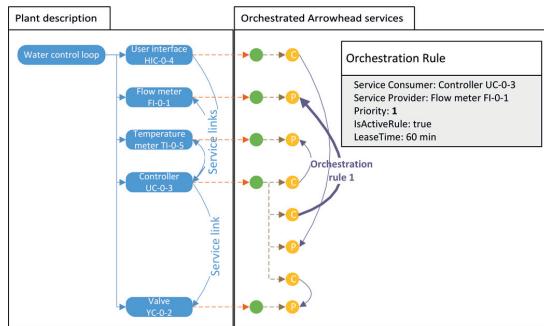


Figure 7: An example for the control elements and their relations, as represented in the Plant Description, and how this information can be used to create orchestration rules. The blue parts illustrate objects in the Plant Description, the green are Systems that can be orchestrated and the yellow are service production and consumption interfaces that are to be connected through service interactions.

Beside giving this representation of the systems and their relations, the Orchestration engine also needs a connection between the nodes in the Plant Description and the running systems requesting orchestration rules. This information could be stored either in the Plant Description, as an additional parameter for each node – by the system itself if the NodeId is given to the system as part of the configuration process –, or in a separate System Registry where additional information about each system is stored as a central repository.

As the Orchestration engine has access to the two pieces of information; i) which system instantiates each node and ii) which nodes should be connected by service interactions, it can compose orchestration rules either on request as systems are connected or as a result of new links being created in the Plant Description as a result of an updated design.

5 Conclusions and Future work

To address a dynamic and changing production environment, it will be important that the configuration of large systems can be changed easily, in a structured manner, according to the intentions of the engineer or designer that initiated the change. The approach presented here shows how the dynamic nature of an automation system based on service interactions can be orchestrated, utilizing standardised engineering data, provided by existing engineering tools.

Additional useful information for the orchestration engine includes the current state of the plant, and how prioritisation can be made based on the current plant state and the current production recipe. Thus future work has to address ways of detecting plant state in run-time, and means of providing service quality information suitable for dynamic orchestration enabling various types of optimisation.

Acknowledgment

The authors would like to extend their gratitude towards EU ARTEMIS JU for funding within project ARTEMIS/0001/2012, JU grant nr. 332987 (ARROWHEAD). We would also like to thank the partners of the Arrowhead project for fruitful collaboration and interesting discussions.

References

- [1] J. Lee, B. Bagheri, and H.-A. Kao, “A cyber-physical systems architecture for industry 4.0-based manufacturing systems,” *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.
- [2] B. Scholten, *The Road to Integration: A Guide to Applying the ISA-95 Standard in Manufacturing*. ISA, 2007.
- [3] S. Karnouskos and A. W. Colombo, “Architecting the next generation of service-based scada/dcs system of systems,” in *Proceedings IECON 2011*, Melbourne, Nov. 2011, p. 6.
- [4] F. Jammes and H. Smit, “Service-oriented paradigms in industrial automation,” *Industrial Informatics, IEEE Transactions on*, vol. 1, no. 1, pp. 62–70, Feb 2005.

- [5] F. Jammes, B. Bony, P. Nappey, A. W. Colombo, J. Delsing, J. Eliasson, R. Kyusakov, S. Karnouskos, P. Stluka, and M. Till, "Technologies for soa-based distributed large scale process monitoring and control systems," in *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2012, pp. 5799–5804.
- [6] G. Candido, A. W. Colombo, J. Barata, and F. Jammes, "Service-oriented infrastructure to support the deployment of evolvable production systems," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 759–767, Nov 2011.
- [7] R. Kyusakov, P. P. Pereira, J. Eliasson, and J. Delsing, "Exip: a framework for embedded web development," *ACM Transactions on the Web (TWEB)*, vol. 8, no. 4, p. 23, 2014.
- [8] J. Delsing, J. Eliasson, R. Kyusakov, A. W. Colombo, F. Jammes, J. Nessaether, S. Karnouskos, and C. Diedrich, "A migration approach towards a soa-based next generation process control and monitoring," in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, Nov 2011, pp. 4472–4477.
- [9] J. Delsing, F. Rosenqvist, O. Carlsson, A. W. Colombo, and T. Bangemann, "Migration of industrial process control systems into service oriented architecture," in *IECON 2012*. IEEE, 2012.
- [10] A. Jain, D. Vera, and R. Harrison, "Virtual commissioning of modular automation systems," in *Intelligent Manufacturing Systems*, vol. 10, no. 1. IFAC, 2010, pp. 72–77.
- [11] N. Kaur, C. S. McLeod, A. Jain, R. Harrison, B. Ahmad, A. W. Colombo, and J. Delsing, "Design and simulation of a soa-based system of systems for automation in the residential sector," in *IEEE ICIT 2013*. IEEE, 2013.
- [12] A. W. Colombo, ed. (2016) Socrates project. [Online]. Available: <http://www.socrates.net>
- [13] 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*, 2014.
- [14] S. Karnouskos, A. W. Colombo, F. Jammes, J. Delsing, and T. Bangemann, "Towards an architecture for service-oriented process monitoring and control," in *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2010, pp. 1385–1391.
- [15] J. Delsing, ed. (2016) Arrowhead framework wiki. [Online]. Available: <http://forge.soa4d.org/arrowhead-f/wiki>
- [16] J. Delsing, ed., *IoT based Automation - made possible by Arrowhead Framework*. CRC Press, Taylor & Francis Group, 2016.

-
- [17] R. Yang, *Process Plant Lifecycle Information Management*. AuthorHouse, 2009. [Online]. Available: <https://books.google.se/books?id=zmgzdf1GR0C>
 - [18] M. Schleipen, R. Drath, and O. Sauer, "The system-independent data exchange format caex for supporting an automatic configuration of a production monitoring and control system," in *2008 IEEE International Symposium on Industrial Electronics*, June 2008, pp. 1786–1791.
 - [19] M. Schleipen and M. Okon, "The CAEX tool suite - User assistance for the use of standardized plant engineering data exchange," in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, Sept 2010, pp. 1–7.
 - [20] R. Drath, A. Luder, J. Peschke, and L. Hundt, "AutomationML - the glue for seamless automation engineering," in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, Sept 2008, pp. 616–623.
 - [21] A. Lüder, N. Schmidt, R. Rosendahl, and M. John, "Integrating different information types within automationml," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Sept 2014, pp. 1–5.
 - [22] T. Holm, L. Christiansen, M. Göring, T. Jäger, and A. Fay, "ISO 15926 vs. IEC 62424 - Comparison of plant structure modeling concepts," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, Sept 2012, pp. 1–8.
 - [23] ISO/IEC 81346-1:2009, *Industrial systems, installations and equipment and industrial products - Structuring principles and reference designations*, ISO/IEC Std.
 - [24] O. Carlsson, D. Vera, J. Delsing, B. Ahmad, and R. Harrison, "Plant descriptions for engineering tool interoperability," 2016, submitted to INDIN 2016 IEEE International Conference on Industrial Informatics.
 - [25] D. D. Clark, C. Partridge, and J. C. Ramming, "A knowledge plane for the Internet," in *SIGCOMM*, Aug 2003, pp. 3–10.
 - [26] C. Pang, V. Vyatkin, and W. Dai, "Iec 61499 based model-driven process control engineering," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Sept 2014, pp. 1–8.

PAPER F

Configuration Service in Cloud based Automation Systems

Authors:

Oscar Carlsson, Pablo Puñal Pereira, Jens Eliasson, Jerker Delsing, Bilal Ahmad, Robert Harrison and Ove Jansson

Reformatted version of paper originally published in:

Conference paper, IEEE IECON 2016, Florence, 2016.

© 2016 IEEE. Reprinted, with permissions, from Pablo Puñal Pereira, Jens Eliasson, Jerker Delsing, Bilal Ahmad, Robert Harrison and Ove Jansson, *Configuration Service in Cloud based Automation Systems*, IEEE IECON, 2016.

Configuration Service in Cloud based Automation Systems

Oscar Carlsson, Pablo Puñal Pereira, Jens Eliasson, Jerker Delsing, Bilal Ahmad,
Robert Harrison and Ove Jansson

Abstract

Current challenges in production automation requires the involvement of new technologies like Internet of Things (IoT), Systems of Systems and local automation clouds. The objective of this paper is to address one of the challenges involved in establishing and managing a cloud based automation system. Three key capabilities have been identified as required to create the expected benefits of local automation clouds; 1) capturing of plant design 2) capturing and distributing configuration and deployment information 3) coordinating information exchange.

This paper addresses the capturing and distribution of configuration and deployment information. For this purpose a system service is proposed, the ConfigurationStore, following the principles of the Arrowhead Framework. The service is accompanied by a deployment methodology and a bootstrapping procedure. These are discussed for several types of automation technology, e.g. controllers, sensors, actuators. A qualitative evaluation of the proposed approach is made for four use cases; Building automation, Manufacturing automation, Process automation and IoT devices. Concluding the usability for large-scale deployment and configuration of Industrial Internet of Things.

1 Introduction

,
High level topics in today's society are sustainability, flexibility, efficiency and competitiveness. These in turn are driven by big societal questions like environmental sustainability, availability of energy and raw material, rapidly changing market trends. We find several trends that in different ways are addressing these topics. One is the move from large monolithic organisations towards multi-stakeholder cooperations where co-operation are fostered by market requirements. Another is the learning from previous products, other parts of the value chain, the life cycle of the product and the product or service production process itself.

These trends are creating new requirements for the technology used to support product and service production, causing a drive for digitisation of production. Digesting this reveals a number of gaps regarding technology, organisation, cooperation structure, operational management and related business models that have to be addressed.

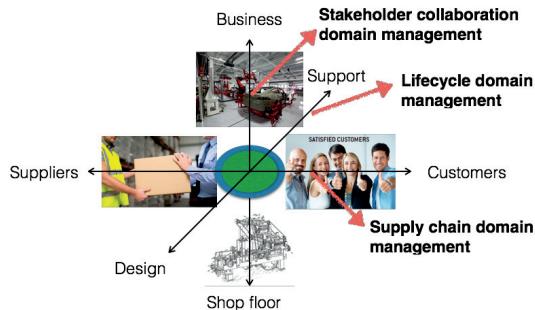


Figure 1: Three important axes for collaborative production, product life cycle, supply chain and stakeholder integration management

Around organisation, cooperation structure and operational management the high profile key aspects are related to three domains, see Figure 1:

- Product life cycle management
- Supply chain management
- Stakeholder integration management

With the move from large monolithic enterprises towards multi-stakeholder cooperation the management is changing towards distributed multi-stakeholder collaboration with distributed responsibilities and decision making. The flexible collaboration along and in-between the three domains also opens for the possibility of dynamic learning. A further aspect is that these domains tend to become wider (longer) involving more stakeholders with diverse objectives and more details and variations of the service or product to meet customer diversity and service and product quality.

These ideas are currently emerging but regarded as very important to address the high level topics of flexibility, efficiency, competitiveness and with suitable incentives also supporting sustainability. To support these developments there are a number of technology gaps which seemingly can not be addressed by current state of the art. Because of this, a number of new technologies are emerging to fill these gaps. Some current big buzz technologies are:

- Internet of Thing, IoT
- System of Systems, SoS
- Cyber-Physical Systems, CPS
- Service Oriented Architecture, SOA

Despite all the new operational and organisational ideas and emerging technologies the automation fundamentals captured by today's state of the art automation technology have to be maintained. Thus, next generation of automation and digitalisation technology has to meet a large set of requirements and involve a wider scope of actors and stakeholders. This is the big challenge for technology suppliers of the future, in this field.

ISA-95, standardised through ISA [1], is today's standard architecture for automation systems [2]. Accompanying the ISA-95 standard are related standards like ISA-99 and IEC 62443 [3, 4], which address control system security.

In 2011, the concept of Industry 4.0 [5] was born in Germany. This concept builds upon the last generation of industrial monitoring and control systems, but enables an even finer level of interaction between shop-floor devices and high-level enterprise systems. In industry 4.0, state of the art technologies like Internet of Things (IoT) and Cyber-physical Systems (CPS) are utilized in order to be able to break the classical, strictly hierarchical, approach of ISA-95 [2] with a more flexible approach without fixed barriers and closed systems.

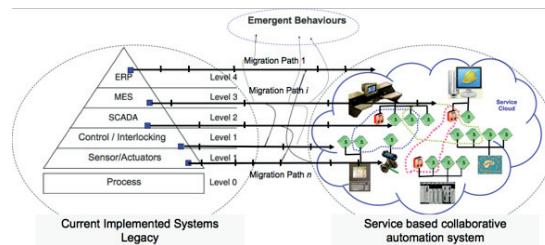


Figure 2: Transferring the ISA-95 automation pyramid architecture to a cloud. This has been investigated by several larger EU projects like e.g. SOCRADES and IMC-AESOP.

The trends and perspectives put forward indicates that the current solutions used to build automation systems are not sufficient. Further the cost connected with the engineering and building of larger automation systems involving multiple stakeholders seems to be prohibitively high.

For the last 10 years, discussion on the next generation SCADA, DCS and MES systems has been around. A multitude of research projects on the topic have been executed. Some more prominent such are SOFIA, SOCRADES [6], and IMC-AESOP [7]. All of them were investigating a move from a hierarchical ISA-95 approach to a more cloud-like approach. A well-known illustration from the IMC-AESOP project is illustrating such move from the pyramid to the cloud in Figure 2. This is currently a rapidly changing landscape but some other efforts touching this field are e.g. FiWare [8]. In addition there are a growing number of cloud offerings on the market. An analysis of the different approaches found in 2015 can be found in [9].

In all these cases the key technology for creating the integration within and in-between different levels of the ISA-95 architecture is Service Oriented Architecture, SOA [10]. SOA

was originally developed by IBM to enable data/information exchange between different lines of computer systems.

For the transfer of the ISA-95 architecture to a cloud based approach we do find some important published work regarding, system architecture [11, 12], suitable technologies[13], real time [14, 15] migration from legacy systems [16, 17], engineering for cloud based automation [18, 19].

A parallel discussion is ongoing for the MES and ERP levels. Some important publications on cloud approaches for the MES and ERP level are [11, 20].

Recently the concept of local automation clouds have been introduced via the Arrowhead project the newly released Arrowhead Framework [21].

Most of the recent developments are adopting Service Oriented Architecture, SOA, as the main approach to enable plant automation using cloud technology.

The objective of this paper is to take a step beyond the current cloud automation technology for production. The ambition is to address the actual process of producing a cloud based automation system. It is here argued that there are at least three important capabilities of the automation clouds which are critical to create expected benefits from local automation cloud approach. The capabilities are:

- A way of capturing a plant design: physical devices, components and systems, geographical layout and controlled interaction between physical devices, components and systems
- A way of capturing and distributing configuration information to the entities involved in the plant design
- A way of coordinating information exchange between different entities within a plant

This paper will address the second point: capturing and distribution of configuration information.

The outline is; Section 2 outlines related work, followed by Sections 3 and 4, which presents the proposed approach and gives examples of a few different use cases with experimental results. Section 5 gives a theoretical discussion on the findings, followed by the conclusions in Section 6. The paper finishes by stating ideas for future work in Section 7.

2 Related work

The configuration has long been a significant part of commissioning automated systems in all parts of society. As different domains and applications form, using different approaches and technology, the configuration procedure has developed in somewhat different directions in different areas.

Hodek and Schlick [22] describe the typical operations for integration of field devices in a state-of-the-art industrial automation system. The device profiles that they

present could very well be used as a standardized baseline configuration for field devices. However, they also note that there are several other components to a quick and simple commissioning, such as the programming of Programmable Logic Controllers (PLC's) and integration into enterprise software systems. Additionally one of the suggested further investigations is on technology independent Plug& Play features of industrial Ethernet solutions.

Cachapa et al. [23] show how an engineering tool based on a Service-Oriented Architecture (SOA) can help by simplifying the process of designing and applying a production line layout, although in their case the configuration still contains manual configuration for each device.

Dürkop et al. [24] present a solution for a reconfigurable automation system where a Programmable Logic Controller (PLC), using Real-time Ethernet (RTE) connected IO-devices, is equipped with a Web-Service Interface that allows configuration of both the PLC and the RTE network.

Perera et al. [25] present a model to help users configure IoT middleware, primarily for middleware used to collect and process data from IoT-enabled sensors. The issue described here illustrates a situation that can be expected to become more common as automation systems become more dynamic and reconfigurable, where users or operators without detailed knowledge in IT or automation are tasked with the reconfiguration of a system and how tools and methods can aid this process.

In conclusion, there are several approaches and solutions that can help improve the engineering and configuration of both existing and future automation systems in many different areas.

3 Proposed Approach

The approach presented in this paper attempts to define certain methods, structures and interaction patterns that will fit in the many areas of society that Arrowhead aims to address. As this will encompass a large number of different technologies and very different external requirements, the general approach is not detailed on a technical level but instead some more detailed scenarios are discussed in the following sections.

3.1 Arrowhead Configuration store

The Configuration Store service is one of the Arrowhead automation support core services.

The purpose of the Configuration Store is to provide a uniform way for Arrowhead compliant system to manage distribution of configurations. The extent of the configurability of a system ultimately depends on the system and therefore the design of this service is intended to allow different levels of configurations to be transferred using the same interface, from changes in system parameters to full firmware updates that may change which services a system is able to produce and consume.

Through this design the decision of how configurable a system should be is left to the system provider, and not imposed by the framework, while still allowing a uniform method for configuration management across diverse systems of systems containing systems with different levels of configurability.

In this design the actual configuration file is not necessarily provided by the Configuration Store, this design was selected to accommodate difference scenarios where accessibility, storage or file transfer ability may otherwise be limiting the distribution of configurations. There is however the possibility to have the same system providing both the Configuration Store service and the appointed file storage.

3.2 General deployment procedure

As the Arrowhead Framework is intended to allow cost effective deployment of a very wide array of devices, all general methods and procedures need to allow for some flexibility and adaptation to the specific use case. Still, many Arrowhead compatible devices are expected to be deployed in large numbers using low-cost hardware manufactured identically in very large numbers. Under these conditions it becomes even more important to keep the costs for engineering, deployment and commissioning at a minimum.

For the general procedure, the devices are assumed to have identical hardware and identical software preloaded from a factory, workshop or back office, the only differences between devices are their network Media Access Control address (MAC address) and their Serial Number (S/N), or some other individual identifier that has been assigned to them automatically during the device manufacturing process. The preloaded software contains the required security measures and to allow the device to connect to the Arrowhead Framework Core systems and to use a minimal set of services. The basic security may, for example, consist of a certificate installed as part of the manufacturing process which certifies that the device is of the brand and type that it claims to be.

To organize the large number of devices into a productive system of systems each device needs to be assigned a specific task and be configured to perform the task according to a larger plan. This information describing the different tasks and configurations is stored in the Configuration Store, as described in section 3.1. In order to allow some flexibility in network structure, without increasing the engineering or deployment time for each device, a specific procedure has been developed.

Step-by-step general deployment procedure:

1. Device is physically connected to the network and turned on.
2. Device connects to the network using DHCP, or a similar standardized technology applicable for the network in question.
3. Device looks for the Arrowhead core systems [26] at predefined locations. (E.g. on the local network or a cloud service hosted by the device supplier)
4. Core systems authenticate device as factory configured device with basic authorization.

5. Human (operator, electrician, engineer or similar) performing the installation associates the physical/logical location (location as registered in the core systems referring to the point where the device is installed) with the MAC address, S/N or other agreed upon identifier of the device.

This can be achieved through a mobile interface (e.g. laptop, smart-phone or tablet) where the installer selects the correct location and either reads the identifier from a bar-code, RFID or similar tag on the device or transfers the identifier from the mobile interface to the installed device using NFC or similar communication.

6. Device registers with the core systems providing its identifier for identification.
7. Once the identifier is available at both the installed device and at the core systems a service connection can be orchestrated between the configurable device and the appropriate configuration store.
8. Once the connection is made between the configuration repository and the identified device, the complete configuration containing the assigned task is transferred to the device and installed/executed.
9. Using the received configuration the device is automatically able to start performing its assigned tasks.
10. A message of success/failure is sent as an event to subscribed user interfaces.

3.3 Bootstrapping of Resource Constrained Devices

As an example of how the general procedure may need to be specialized to fit certain requirements, a more limited procedure has been designed and tested. The purpose of this is to show how a general procedure may be of use even though the application field is very diverse and it may be difficult to apply the original procedure explicitly in every case.

A zero configuration approach for IoT devices requires the use of Bootstrapping techniques. From the point of view of a wireless sensor and actuator (WSAN) node, the first time that a new node connects to the wireless network it only knows its own IP address and the IP address of the gateway; it has normally no information about other services in the network.

Bootstrapping is a pseudo-configuration service which provides a basic configuration of the mandatory and essential services that a node requires and needs during the boot process, examples are; configuration services, authentication services or device manager service.

As the only information that a node has after connect to the wireless network is the IP of the gateway, the Bootstrapping service must run on it and should use a predefined port (this is the unique predefined information). The bootstrapping request can include information about the IoT device to create a customized and optimized response for the device; this information can contain a serial number, a predefined ID, MAC address,

internal software name, version, or other information able to identify the device or at least the device type.

After this process, the device should store all the information in a non-volatile memory, and use it in case the connection to the bootstrapping service goes down.

The penalty for the utilization of this technology regarding communication is a single request per boot but also it requires the implementation of a parser on the device, which can consume valuable memory on resource-constrained devices.

The following example is a bootstrapping profile encoded with JSON (Code F.1), but it can be encoded with CBOR to reduce the packet size.

Code F.1: Example of Bootstrapping for an IoT node encoded in JSON

```
{
  "auth": {
    "ip": "fdfd:0:0:0:0:0:0:0A",
    "port": 5683,
    "v": 1,
    "res": "/Authentication",
    "resAlt": "/Authorization"
  },
  "conf": {
    "ip": "fdfd:0:0:0:0:0:0:0B",
    "port": 5682,
    "v": 1,
    "res": "/Conf"
  },
  "dev": {
    "ip": "fdfd:0:0:0:0:0:0:0C",
    "port": 5681,
    "v": 1,
    "res": "/rd"
  }
}
```

3.4 Intended areas of configuration

To illustrated how the configuration approach may provide different possibilities for different areas of application, this section provides a few examples from some of the domains where Arrowhead is intended to be used.

Control code to PLC's and IoT controllers

Writing control code is one of the critical aspects of automation systems' engineering process. The traditional approach to program control systems does not fit well within the paradigm of Cyber-Physical Systems (CPS), where physical systems need to evolve in intimate and aligned correspondence with their virtual representation [27]. To address this, the Arrowhead approach propose a data-driven method for control code deployment, deployable on a number of devices and platforms with embedded data storage and processing capabilities, for the configuration of automated manufacturing systems.

The proposed architecture is composed of three types of elements (see Figure 3): i) data model that describes the system structure and the control behaviour, ii) logic engine that orchestrate system by interpreting description of the system defined within the data model, and iii) resource specific standard library Function Blocks (FBs) acting as an interface between the hardware (i.e. sensors and actuators) and the data model.

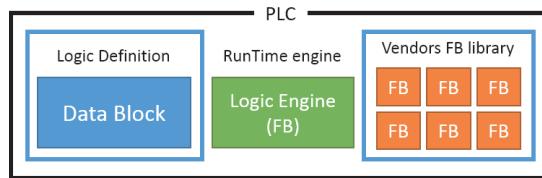


Figure 3: Control software architecture

In this software architecture, the logic engine is configuration independent as for any system configuration it remains unchanged while the data model is configuration dependent as any change in the system, such as sequence, require reconfiguration of the data model. Resource-specific FBs can remain the same for different control or hardware configuration, but may need to be changed in some cases (e.g. vendor specific configuration, actuator type/configuration etc.).

The motivation for this software architecture is to dissociate the key elements required to achieve the overall device configuration and therefore dissociate the engineering processes that support various aspect of the device configuration (e.g. device firmware/logic engine update, system specific configuration change/update, etc.). This approach also allows generating the control code using standard library components (i.e. FBs), which are driven by the control logic defined in the manufacturing process simulation tools to enable seamless transition from virtual to physical system and feedback of runtime data to virtual system to facilitate data model calibration and analytics. As the control behaviour is defined in the data model, which can be accessed (i.e. read/write) in runtime, service visualization and process parametrisation can be attained using human-machine interface devices.

Configuration of sensors and actuators

Sensor and actuator nodes usually have two types of physical resources: sensors and actuators; and in the case of IoT systems based on Service Oriented Architecture (SOA) the configuration can also set the service's behaviour. Therefore, there are three different types of configurations.

1. Sensor

- Sample rate

- Inactive periods
 - Sensitivity
2. Actuator
 - Sensitivity
 - Active/Sleep
 3. Service
 - Service composition
 - Filtering
 - Data compression
 - Output format
 - Triggering

Configuration of Building Automation Controllers

Building Automation Controllers (BAC's) use multiple physical resources: sensors, actuators and remote i/o units as well as being freely programmable with applications for monitor & control, communication, HMI and event handling. Therefore, there are many different types of configuration but they can be divided into functional configurations (programming, setup, services) and operational configurations (settings), the later often provided by services in a SOA environment.

4 Use cases and Evaluation

The proposed approach is intended to be flexible and to provide different possibilities for scenarios likely to be encountered in different areas of automation. Some use cases have been collected to illustrate the benefits, and possible drawbacks of the approach.

4.1 Use case: Building automation

In the field of building automation it is common that many systems in one area are to use identical configurations, or that there are a few typical configurations that are used for a large number of systems. This may be the case for an area with apartment buildings that are all built during the same era and are owned by the same company, here it is beneficial for the owners from a maintenance and management point of view if the systems are as similar as possible.

In this case, and similar scenarios, a centralised Configuration store allows management and updates of all systems based on one configuration that can be rigorously tested and monitored for its first deployment. Once the initial deployment has proven successful it can be applied to all others with low risk of failure.

Additionally, a Configuration store that can keep track of configuration status will allow easier comparison of similar buildings using different configurations, in order to optimise or find flaws in the tested versions.

4.2 Use case: Manufacturing industry

Due to the growing need to manufacture highly innovative and customized products, efficient and quick adaptation of manufacturing systems to new product and production volumes is of significant importance. It has been established that one of the major obstacles in realising an efficient and reconfigurable production systems is the existing PLC control code development and deployment approach. The management of PLC devices configuration is currently completely dissociated from other engineering phases (such as process planning and mechanical engineering) and from the digital data set resulting from them.

WMG and FDS is developing a virtual engineering toolset, vueOne, and associated CPS oriented engineering methods that aim at filling this gap by providing data-driven control code generation capabilities (described in section III, D) directly from the vueOne virtual process planning module [27]. In this use case, an engineering scenario focusing on PLC devices configuration is investigated using Web service and Arrowhead Framework to enable direct deployment of control software to PLC devices to provide basis for dynamic and more distinctive configuration scenarios.

Unlike the classical method of PLC device configuration, which requires a direct connection between the PLC and a laptop running the vendor-specific programming tool, the PLC devices (or a server module linked to it) subscribes to the Configuration Store. Any changes in the control configurations are then directly passed from the Configuration Store to the subscribed PLC device when available. To capture the changes made to in the control code on the shop-floor, if any local changes are made in the control software, such as changing some parameters of the data model, the PLC device uploads the latest configuration to the Configuration Store to update the configuration database.

In an ideal configuration mechanism, the PLC device will retrieve updated configuration if/when available automatically. In practice however, the approach requires access to proprietary APIs from PLC vendors to allow download access to the PLCs. The Configuration Store consumer software is currently deployed on laptops that control engineers connect to PLCs in order to update or install control code. However, in case of embedded controllers the dynamic configuration can be achieved seamlessly using the Arrowhead framework due to their non-proprietary configuration mechanisms.

4.3 Use case: IoT devices

This section is based on experimental results of energy consumption and delays. The benchmark configuration relies on measures of battery current and voltage externally to the device; these measurements are done using a 16-bit ADC at 1840 Hz to capture rapid events such as radio, wakeups, etc. All these measurements are combined to 8 digital

inputs that can be used to recognize in detail the power consumption of each software module.

The selected IoT platform to do the test was a Mulle from Eistec AB, which is equipped with an ARM Cortex-M4 at 100 MHz microcontroller and an IEEE 802.15.4 transceiver. It has an onboard 2 MB of flash memory and 256 KB of internal memory on the microcontroller. The Mulle runs the open-source Contiki OS; so all taken measures are affected by running an OS on the same device without any isolation to get real condition data.

As the number of devices increase, so does the need for technologies for large scale management of systems including hardware devices and life cycle management. The issue of life cycle management, in particular configuration, is even more complicated when it comes to managing a very large number of resource constrained, wireless and battery powered devices in harsh environments.

In this use case, the authors have investigated how advanced configuration can be achieved in a very efficient manner in terms of communication overhead and energy usage.

The introduction of the IP technology for Wireless Sensor Networks (WSNs), now called Internet of Things (IoT) is today a hot topic, the power consumption of application's level protocols was a barrier which hindered a massive expansion of IoT; But in 2014, the standardization of CoAP[28] helped to develop IoT systems. The application of CoAP was a step forward and changed the behaviour of the WSAN nodes, from simplistic pull-based to more complex event-based communication. Nowadays, each node can provide services (resources) to other nodes, or to any server/client at Internet. This enables deployment of smart and efficient IoT systems with advanced features such as; service composition, event detection, low-power operation, high dependability, etc.

All these new features requires either run-time zero configuration and/or static configuration; In order to provide run-time configuration which is a much better approach for deploying larger networks, the framework must implement both bootstrapping and configuration services. The validation of these services for bootstrapping and configuration are a direct comparison between the benefits they provide versus their respective performance impact in terms of power consumption, communication overhead and memory usage.

The proposed Bootstrapping and Configuration services enable the following features:

- Low-power. The configuration of the previous WSNs was static; that means that the performance of each device was the same during the life-cycle. But now with Configuration service, the performance can be adaptive. All variables like sample rate, type of processing, triggers, etc. can be modified.
- Dependability and Robustness. The bootstrapping nature is dynamic, on each request, the bootstrapping service creates a customized response; Then if a service goes down, another one can replace it, just changing the bootstrapping parameters.
- Zero Configuration. With bootstrapping there are no fixed end-points on the device,

one node could be deployed at any WSNs, and it will be able to establish a correct configuration.

The consumption of these two services (Bootstrapping and Configuration) is not excessive compared to others as Authentication, Authorization and Device Manager. The values at Figure 4 can change depending on the complexity of the device configuration.

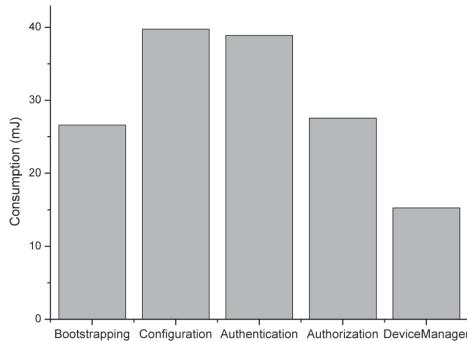


Figure 4: Comparison of energy consumption between Bootstrapping, Configuration and other common IoT-WSNs services.

Regarding memory, the biggest cost is to parse the incoming configuration profile. For this experiment, the profile was encoded in JSON format, and the jsmn[29] C-API was used as the parser. The configuration memory usage represents the memory of configuring a single service. The bootstrapping memory usage also includes the JSON parser's ROM memory usage (see Table 1).

	Bootstrapping	Configuration
ROM (bytes)	1126+4382(parser)	840
RAM (bytes)	412	776

Table 1: Memory footprint of Bootstrapping and Configuration

4.4 Use case: Process industries

The process industry typically has comparatively static production lines where reconfigurability of the production, as commonly described for manufacturing industries, is not usually sought after. However, another scenario that may require significant configuration is that of replacement of devices, machines or groups of systems.

In the current environment of process industry automation systems, replacement of devices or larger systems generally takes place because of one of two reasons.

The first reason is that the existing one is broken, faulty or worn out and should be replaced with a new, but identical, item. This case is generally considered to be part of standard maintenance procedures, either reactive or proactive.

The second reason is that the existing item is too old and should be replaced with a newer part, this may be due to e.g. discontinued support from suppliers and difficulty to procure replacements, difficulty to attract and retain personnel with the required expertise in older systems or due to lack of technical features or functionality. This case is generally considered part of the long term investments in production systems and as such, replacements are generally planned carefully and in great detail.

While the conditions before the two cases are very different, especially with regards to the possibility to make preparations, as one is planned and the other is not, there are also some commonalities in that in both cases the functionality is expected to be exactly the same as before and that the time for the replacement and commissioning is usually very limited.

Traditionally these kind of replacement procedures have relied heavily on existing documentation and backup copies of control code and configurations, however these are not always kept up to date and may not be fully compatible with updated replacement devices. This leads to additional engineering work and may in unfortunate cases lead to unforeseen complications during the critical commissioning work.

As the Configuration store can be used to store the most current configuration for each device and the deployment procedure allows for a smooth introduction of new devices, an easier, quicker and cheaper replacement process is expected.

However, the Configuration store does not solve the issue with incompatible configurations or software version, but having a backup that is certainly of the most recent configuration will make engineering work easier and the re-commissioning less uncertain. If the device manufacturers can be persuaded to use open and standardized configuration files this would allow more possibilities for conversion and testing of updates ahead of the physical replacement.

5 Discussion

The use cases presented are diverse and have significantly different requirements and potential benefits. The case presented from Building automation highlights the benefits from a management and maintenance point of view. The use case from a Manufacturing industry illustrates some of the strong potential once an approach like this is fully integrated in a complete engineering tool chain, something that can be expected to happen eventually in all areas with significant engineering requirements. However, it also illustrates some of the limitations that may occur when existing hardware is not yet capable to make use of new possibilities.

The IoT case illustrates the drawbacks that can be expected from using the approach for resource constrained devices. The cost incurred in terms of energy and memory

consumption is not negligible, but for scenarios where a large number of devices are to be deployed and configured it is likely to be acceptable. At least when compared to the additional engineering time required to configure all devices individually prior to deployment. The use case from a Process industry shows that a structured, centralised management of device configurations can provide benefits to diverse elements of society. In addition, keeping the records of device configurations automatically updated is likely to remove some of the additional cost incurred when a device needs to be replaced or upgraded.

6 Conclusion

In this paper, we have presented the Arrowhead Framework's approach for advanced configuration of systems and devices. The proposed approach is designed to be highly versatile while still being efficient for usage by resource-constrained devices. The approach has been implemented and tested on a resource-constrained wireless sensor and actuator platform.

Test results indicates that the overhead from performing bootstrapping and automatic configuration of a newly deployed device is negligible in terms of energy consumption and memory usage in relation to the energy spent by the device for sensing purposes during its lifetime. This shows that advanced run-time configuration is feasible even on small, battery-powered devices.

7 Future work

Among the possible paths of future advances can be found several interesting options. Along the path of further implementation lies the tasks of prototype implementations for configuration of larger, less limited devices.

Further development could include implementations more integrated with engineering tools. This could be done either by implementing methods for managing the ConfigurationStore from existing, discipline related engineering tools, or by designing an engineering tool centred around the ConfigurationStore and associated processes. The former implementation would allow for a smooth engineering and deployment process within the specific domain, while the latter has an advantage of spanning all affected domains easing e.g. management and maintenance of multi-domain facilities.

Acknowledgment

The authors would like to extend their gratitude towards EU ARTEMIS JU for funding within project ARTEMIS/0001/2012, JU grant nr. 332987 (ARROWHEAD). We would also like to thank the partners of the Arrowhead project for fruitful collaboration and interesting discussions.

References

- [1] (2016). [Online]. Available: <https://www.isa.org>
- [2] B. Scholten, *The Road to Integration: A Guide to Applying the ISA-95 Standard in Manufacturing*. ISA, 2007.
- [3] (2016). [Online]. Available: <http://isa99.isa.org/ISA99%20Wiki/Home.aspx>
- [4] K. Staggs and et.al., “ISA 62443 – 4 – 2 security for industrial automation and control systems technical security requirements for iacs components,” ISA, Draft Standard Draft 2 edit 4, July 2015.
- [5] J. Lee, B. Bagheri, and H.-A. Kao, “A cyber-physical systems architecture for industry 4.0-based manufacturing systems,” *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.
- [6] A. W. Colombo, ed. (2016) Socrades project. [Online]. Available: <http://www.socrades.net>
- [7] 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*, 2014.
- [8] Wikipedia, “FiWare — wikipedia, the free encyclopedia,” 2016, [Online; accessed 21-April-2016]. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=FIWARE&oldid=711836994>
- [9] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller, “A survey of commercial frameworks for the internet of things,” in *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*. IEEE, 2015, pp. 1–8.
- [10] T. Erl, *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007.
- [11] S. Karnouskos and A. W. Colombo, “Architecting the next generation of service-based scada/dcs system of systems,” in *Proceedings IECON 2011*, Melbourne, Nov. 2011, p. 6.
- [12] F. Jammes and H. Smit, “Service-oriented paradigms in industrial automation,” *Industrial Informatics, IEEE Transactions on*, vol. 1, no. 1, pp. 62–70, Feb 2005.
- [13] F. Jammes, B. Bony, P. Nappey, A. W. Colombo, J. Delsing, J. Eliasson, R. Kyusakov, S. Karnouskos, P. Stluka, and M. Till, “Technologies for soa-based distributed large scale process monitoring and control systems,” in *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2012, pp. 5799–5804.

- [14] G. Candido, A. W. Colombo, J. Barata, and F. Jammes, "Service-oriented infrastructure to support the deployment of evolvable production systems," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 759–767, Nov 2011.
- [15] R. Kyusakov, P. P. Pereira, J. Eliasson, and J. Delsing, "Exip: a framework for embedded web development," *ACM Transactions on the Web (TWEB)*, vol. 8, no. 4, p. 23, 2014.
- [16] J. Delsing, J. Eliasson, R. Kyusakov, A. W. Colombo, F. Jammes, J. Nessaether, S. Karnouskos, and C. Diedrich, "A migration approach towards a soa-based next generation process control and monitoring," in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, Nov 2011, pp. 4472–4477.
- [17] J. Delsing, F. Rosenqvist, O. Carlsson, A. W. Colombo, and T. Bangemann, "Migration of industrial process control systems into service oriented architecture," in *IECON 2012*. IEEE, 2012.
- [18] A. Jain, D. Vera, and R. Harrison, "Virtual commissioning of modular automation systems," in *Intelligent Manufacturing Systems*, vol. 10, no. 1. IFAC, 2010, pp. 72–77.
- [19] N. Kaur, C. S. McLeod, A. Jain, R. Harrison, B. Ahmad, A. W. Colombo, and J. Delsing, "Design and simulation of a soa-based system of systems for automation in the residential sector," in *IEEE ICIT 2013*. IEEE, 2013.
- [20] S. Karnouskos, A. W. Colombo, F. Jammes, J. Delsing, and T. Bangemann, "Towards an architecture for service-oriented process monitoring and control," in *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2010, pp. 1385–1391.
- [21] J. Delsing, ed. (2016) Arrowhead framework wiki. [Online]. Available: <http://forge.soa4d.org/arrowhead-f/wiki>
- [22] S. Hodek and J. Schlick, "Ad hoc field device integration using device profiles, concepts for automated configuration and web service technologies: Plug and play field device integration concepts for industrial production processes," in *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*, March 2012, pp. 1–6.
- [23] D. Cachapa, R. Harrison, and A. Colombo, "Configuration of SoA-based devices in virtual production cells," *International Journal of Production Research*, vol. Volume 49, no. Number 24, pp. 7397–7423, 2011. [Online]. Available: <http://wrap.warwick.ac.uk/54497/>
- [24] L. Dürkop, H. Trsek, J. Otto, and J. Jasperneite, "A field level architecture for reconfigurable real-time automation systems," in *Factory Communication Systems (WFCS), 2014 10th IEEE Workshop on*, May 2014, pp. 1–10.

- [25] C. Perera, A. Zaslavsky, M. Compton, P. Christen, and D. Georgakopoulos, “Semantic-driven configuration of internet of things middleware,” in *Semantics, Knowledge and Grids (SKG), 2013 Ninth International Conference on*, Oct 2013, pp. 66–73.
- [26] F. Blomstedt, L. L. Ferreira, M. Klisics, C. Chrysoulas, I. M. de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, “The arrowhead approach for soa application development and documentation,” in *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, Oct 2014, pp. 2631–2637.
- [27] R. Harrison, D. Vera, and B. Ahmad, “Engineering Methods and Tools for Cyber-Physical Automation Systems,” *Proceedings of the IEEE*, vol. 104, no. 5, pp. 973–985, May 2016.
- [28] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP),” RFC 7252 (Proposed Standard), Internet Engineering Task Force, Jun. 2014. [Online]. Available: <http://www.ietf.org/rfc/rfc7252.txt>
- [29] “Jsmn C JSON parser,” <https://github.com/zserge/jsonn>, updated: 2016-01-15.

PAPER G

Engineering of Service-oriented IoT Automation Systems

Authors:

Oscar Carlsson and Jerker Delsing

Reformatted version of paper submitted to:

Journal paper, IEEE Systems Journal, 2017.

© 2017 IEEE. Reprinted, with permissions, from Jerker Delsing *Engineering of Service-oriented IoT Automation Systems*, IEEE Systems Journal, 2017.

Engineering of Service-oriented IoT Automation Systems

Oscar Carlsson and Jerker Delsing

Abstract

Cyber-Physical Systems and Internet of Things are envisioned to revolutionise automation systems in the near future and Service-Oriented Architectures are heralded as an enabling technology. To allow the new technology to contribute to automation solutions, at the scale envisioned, there is a need for an efficient way to organise the hardware and software to fulfill useful functionality in line with the intentions of a designer.

This paper presents an engineering work-flow, composed of certain concepts that are perceived as critical for efficient design, engineering and deployment of automation systems based on Internet of Things. The work-flow is presented using concepts from the Arrowhead Framework, but should be applicable for any similar solution.

The presented work-flow is illustrated through application in a automation scenario where it can be compared to the engineering of a traditional automation solution. The qualitative analysis indicates a decreased effort for certain tasks, without any identified situations where the effort is significantly increased.

1 Introduction

In the coming years, concepts such as Industrie 4.0 [1], Cyber-Physical Systems (CPS) and Internet of Things (IoT) are posed to drastically change many aspects of industrial automation. These concepts, and others, are to be utilized to break down the strict traditional ISA-95 [2] hierarchies and replace them with a more flexible approach with barriers between systems based on requirements and design goals, rather than technical limitations or traditional concepts.

For the interaction between Cyber-Physical Systems, the connected *Things*, Industrial Agents and other types of systems, Service-Oriented Architectures (SOA) have been heralded as an enabling technology.

Rooted in the Arrowhead project [3], the effort described here was initiated by the following basic research questions:

- How should a user involved in the design, engineering, operation or maintenance of an Arrowhead System-of-Systems (SoS) be able to find the correct device or system, given that an Arrowhead SoS is anticipated to span many different domains and that different users are used to different ways of finding a device or system, depending on their expertise and background?

- Can a system providing this functionality be put to further use as an Arrowhead Core system?

Conceptually, the introduction of new automated systems in most domains, and especially in industrial applications, go through five steps from idea to operation, as illustrated in Figure 1:

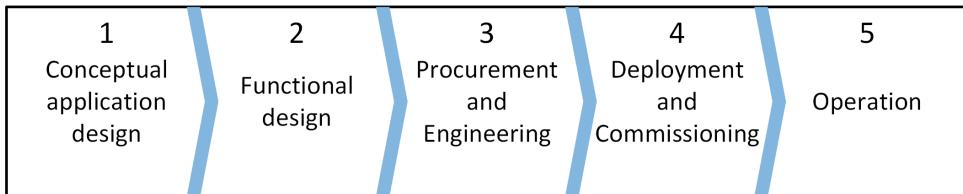


Figure 1: Five steps of an automation system engineering work-flow.

- 1. Conceptual application design:** This step is comparatively informal and usually not technically specific but should outline the purpose and motivation for the application. Typically it should answer the question “**Why?**” and provide basic requirements to the design, given available resources and other constraints.
- 2. Functional design:** This step will produce most of the overarching design documents. The results of this should be a detailed design of the mission critical functionality and functional requirements for the support systems.
- 3. Procurement and Engineering:** These two activities are often performed in parallel, and will affect each other. Certain requirements and design decisions may limit procurement to a single option, that will affect engineering. Subsequently, engineering decisions may pose additional requirements on procurement of other parts or subsystems.
- 4. Deployment and Commissioning:** As all equipment is purchased, configured for the application, and delivered to the site of operation, a process of deployment and integration of all systems begin. When systems have been connected a series of integration tests and commissioning starts, after which the full, interconnected system-of-systems is ready for operation by the end user.
- 5. Operation:** When the operational phase starts, the full system-of-systems *should* perform at the requested level without any need for further engineering. However, in most scenarios there are minor details that were not foreseen during the design phases and must be adjusted. Similarly, external factors, such as markets or regulation, are likely to change during the operation of a large system. All of these may require updates to design and engineering, including documentation and data.

One of the key selling points for IIoT-systems has been the increased flexibility, incurring benefits in the operational phase and, at least initially, the introduction of new technology is not expected to fundamentally change the desired application, the “Why?” of the system. Therefore, the scope of this work has been primarily on the tasks included in **Functional design, Procurement and Engineering**, with some inclusion of the output of the previous step and the explicit goal of facilitating **Deployment and Commissioning**.

This paper is intended to present a general engineering work-flow for IoT automation systems, and to discuss the benefits of this work-flow when applied to an IoT framework, qualitatively compared to using traditional engineering procedures for an automation system using a Programmable Logic Controller (PLC) for the control loops and a Supervisor Control And Data Acquisition (SCADA), for the two bottom levels of the ISA-95 architecture [4].

The outline of the following sections starts with a brief overview of other significant results in the area, summarised in Section 2. Section 3 contains a description of what is expected from the IoT automation framework and a summary of the Arrowhead Framework [5] which is used for the following sections to describe the engineering work-flow. In Section 4, the Engineering work-flow is described. Section 5 describes an automation scenario and Section 6 illustrates how the Engineering work-flow can be applied in this scenario, with some reflections on similarities and differences compared to engineering of traditional automation systems. Section 7 summarises the qualitative analysis with some significant observations and the paper ends with some suggestions on possible future research areas in Section 8.

2 Related work

For traditional process plants, Yang [6] describes the engineering as divided into two phases. The first phase consists of Process and Front-end engineering to determine the process, i.e. what product is to be produced in the plant, how is it to be produced and how much of it is to be produced. This is to a large extent captured by the first two steps of the engineering work-flow described in Section 1, Conceptual application design and Functional design. The second engineering phase described by Yang is called Detailed engineering and correlates to the Engineering included in the third step of the work-flow in this paper.

In the life-cycle described by Yang, the engineering is followed by Procurement, Construction and finally Operation/Maintenance. The engineering work-flow presented in Section 1 comes from the same tradition as the one described by Yang, but has been modified to better accommodate different application domains and focus more on the engineering of automation systems.

For traditional industrial automation systems there are already established engineering work-flows, with a heritage from the requirements for rigorous documentation, sometimes referred to as Document-Centric Engineering. As Scheeren and Pereira [7] demonstrate, there are significant advantages to Model-Based Systems Engineering (MBSE),

but as Logan et al. [8] illustrate the document-centric approach does still have its points, and the two must not be entirely exclusive:

“Whether the process is based on or centred in the model, documents are central to developing, verifying, validating and making use of the model” [8].

The engineering work-flow presented in this paper is intended to be possible to integrate with Document-Centric Engineering, based on existing CAx-tools, as well as model-based methods. Compared to the Model-Based Systems Engineering (MBSE), the Document-Centric approach has a significant advantage in that the traditionally schooled engineers are most likely more used to CAx-tools than to modeling tools.

For the purpose of data exchange between engineering environments of different engineering domains Moser and Biffl have presented the engineering-knowledge-base (EKB) [9, 10]. Their work is focused on mapping of data between engineering tools using ontologies. The engineering-knowledge-base is part of the concept for an Engineering Service Bus (EngSB), which was initially introduced by Biffl and Schatten for software engineering [11], but has also been proposed as part of the Automation Service Bus (ASB) introduced by Biffl et al. [12].

The engineering work-flow presented here is composed using the technical approaches discussed by Carlsson et al. [13, 14, 15].

3 IoT automation architecture

The engineering work-flow presented in the following sections is intended to reflect the work required to bring an IoT automation system from the point where a functional design has been developed to where the automated system is operational.

To provide a qualitative analysis of the proposed engineering work-flow, some clarifications should be made regarding the architecture for the IoT automation systems to be engineered.

It is assumed that the *Things* of these IoT automation systems are interoperable, based on a Service-Oriented Architecture (SOA), interacting through the consumption of services provided by other systems, and that these services are abstracting underlying communication infrastructure to the point that any two provided services of the same type and similar enough specification are technically interchangeable.

For the management of service interactions between the *things* it is assumed that there are one or more entities acting as orchestrators of the whole system-of-systems, with the responsibility to convey the intentions of the application designers to have certain *things* cooperating. Furthermore, there is a need for one or more trusted entities to act as guarantors toward connected *things*, allowing the *things* to trust each other. Additionally, there is a need to maintain the information of what systems are available and what services they provide.

If the *things* have more than one mode of operation or can be configured for different scenarios, it is important to have an efficient approach to configuration management. It is therefore assumed that the architecture contains a coherent method of administrating the configuration of *things*.

For automation scenarios, there are often situations where the operation of the larger system-of-systems is dependant on response times or latencies, sometimes referred to as “real-time requirements”. For such situations it is assumed that the architecture contains or enables options to measure, control, simulate or predict such parameters, both on an individual system level and on a system-of-systems level.

Additionally, some information is assumed to be available with functional aspects of each service, considered as meta-data for the service.

For example, IoT automation systems developed using the Arrowhead Framework [5] is one class of such automation systems. Within the Arrowhead Framework, the necessary management and assurance mentioned above has been implemented as a number of systems to support the connected *things*, these systems are collectively referred to as Arrowhead Framework Core Systems and are described by Varga et al. [16].

As such, Arrowhead Framework compliant IoT's will subsequently be used when exemplifying the engineering work-flow discussed and therefore a number of Arrowhead Framework names for the concepts mentioned above are used in the description and discussion. Most important are:

- **Local Automation Cloud:** Some of the challenges with regards to scalability, security and performance have in the Arrowhead Framework been solved using the concept of a Local Automation Cloud, as described by Delsing et al. [5, 17].
- **System and Device:** An Arrowhead Framework system is primarily intended as the software entity providing and consuming Arrowhead compliant services. A distinction is made towards the hardware upon which the system is executed, which is considered an Arrowhead compliant device. A device may host more than one system [5].
- **Application and Core systems:** Within the Arrowhead Framework a distinction is made between the Application systems that are installed to perform an automation function for a functional application scenario and the Core systems that are scenario independent and intended to support and manage the application systems [5].
- **Orchestrator:** The Orchestrator or Orchestration system is one of the Core systems, responsible to assign service providers to consuming systems. Several implementations have been made, as described by Hegedűs et al. [18], but with regards to the engineering work-flow these are interchangeable.
- **Authorisation system:** The core system used to manage information assurance aspects such as authorisation, authentication and accountability is most often called Authorisation system [5].
- **Device, System and Service registry:** The Service registry is one of the mandatory Core systems and is where all systems are responsible to register services made available for consumption. The service registry may be used by the orchestrator

for look up of available services. The Device registry and the System registry are a non-mandatory Core support systems, designed to facilitate device and system management [5].

- **Configuration system:** For a generic approach to management of configurable devices and systems, the Configuration system provides a service acting as an organised storage for configuration files [15].
- **Plant Description:** To allow an engineer or operator to manage the Local Automation Cloud and to capture the intended automation design, the Arrowhead Framework contains the Plant Description. The Plant Description concept [13], is intended to act as an intermediate abstraction layer, based on existing standards for design and engineering data, isolating only the objects and their relations. One key usage of the Plant Description is as a source of design data for the orchestrator [14].
- **Quality of Service:** In order to provide reliable performance in a Local Automation Cloud the Arrowhead Framework contains a concept built on monitoring and calculating network performance, and subsequently limiting new orchestrations. The system responsible for such monitoring and calculations is referred to as the Quality of Service system (QoS-system) [19].

4 IoT automation engineering

Starting from the beginning of the engineering work-flow outlined in Section 1, the process of Conceptual application design is not expected to be altered due to the new possibilities introduced by the Internet of Things, as the process is essentially non-technical. New applications may arise and the designs produced may utilise the new opportunities, but the process will most likely remain very similar.

4.1 Functional design

The functional design is assumed to provide a set of interacting objects, providing different functions in the overall design. This design should contain all required information *required* for an automation system of systems, when combined with all applicable standards and regulations.

Similarly to the conceptual application design, the functional design process is not expected to change much due to the introduction new technology. The major foreseen changes is that there is a need to add certain design aspects to the tools and standards used in the process. For example, the flexible nature of IoT that allows for late binding and loose coupling among systems is traditionally not captured in the design documents. Traditionally, connections e.g. between a sensor, an actuator and the controller in a control loop are strictly defined design documents. If the design tools and standards are be amended to allow for e.g. some flexibility or prioritisation in the designed connections, this could later be implemented as alternative orchestrations between the systems.

The expected output of the Functional design can be summarised as a set of design documents, traditionally with the purpose to collect the requirements and design decisions in a form that can be understood by engineers with minimal risk for misinterpretation. Often the documents consist of technical drawings of the overall physical layout, some drawings or flow-charts of the automated process, a list of identified technical components and required parameters, and a textual description of the process that is to be automated.

4.2 Procurement and Engineering

The ultimate purpose of the steps following the functional design is to convert the provided information into something that the hardware and software can execute, and to document it so that it is accessible to all the people that will interact with the systems during their complete life-cycle. From the provided design documents, applicable standards, and the financial limitations of the project, it is up to the engineers to select the most suitable hardware and software, and to provide all of the details required to build and document the working system.

Throughout the engineering work-flow, engineers of all involved disciplines will be refining the information captured in the design documents into more detailed drawings and “engineering data”, as the specification of one component often lead to further requirements on other components, sometimes between engineering disciplines. Ideally, there are prepared design sections, or “typicals”, that are known to work well together. These may be put together by system suppliers, end users or by engineering specialists, from previous similar projects, and can be of use to engineers of all disciplines.

In the context of the Arrowhead Framework, it can be assumed that each individual *Device* or *System* can be seen as one of these building blocks, with associated “typicals” complementing the service and system descriptions described in [20]. Due to the interdisciplinary work carried out in this phase of the engineering work-flow, it is essential that identities of each object, system and component are clear to all involved disciplines.

System-of-Systems Management and System Configuration

Traditionally, the design-time management of systems is considered part of each associated engineering discipline, possibly with a common system structure, but often not. In addition, any relationship between systems not captured as part of the system structures must then be captured through specific parameters or additional references in the engineering data.

The proposed Plant Description concept [13, 14], is intended to act as intermediate abstraction layer, based on existing standards for design and engineering data, isolating only the objects and their relations. The reasons for this approach, rather than either selecting one existing standard or contributing to the efforts to harmonise groups of standards, are twofold.

Primarily, the approach was designed to be applicable in as many application domains as possible, this results in a large number of existing standards and different opinions

on what data is the most important. Secondarily, given a highly modular framework with independent systems interacting through services interactions, there are many use cases for a very limited set of the engineering data and much of this can be abstracted to identification of the systems and their respective relations and interaction patterns.

For an engineering scenario where systems are more interchangeable and can to a larger extent be considered black or grey boxes, this abstraction helps engineers to focus on the more important interactions and exchanges between the systems as the internal workings of the system is assured by the system providers. Some examples of this will be discussed in Section 5.

The concept of a Configuration Store service [15] was conceived as part the same approach to system management, in the that the management of the configurations for each systems was generalised to the point where only one user interface is needed to assign configurations to systems from many different system providers.

The scope of configuration is by and large decided by system providers, based on reasons spanning from physical and technical limitations to market strategies and company policies. Given the diversity of fairly standardised configuration strategies even in relatively closely related domains such as industry [21], building automation [22], and substation automation [23] it does not seem likely that there will be an agreement on the format of configuration files in the near future. In this situation, the Configuration Store concept is proposed as a lowest common denominator, with the possibility of further extensions into fully standardised configurations for willing system providers.

Together with the deployment procedure, the Plant Description and the Configuration Store is expected to reduced the engineering time and effort of compliant IoT automation systems across all application domains. An example of how these concepts work together is given in Section 6.

4.3 Deployment and Commissioning

Given the innate local aspect of automation systems there are some limitations that should be considered. With the possibility of having Local Automation Clouds completely disconnected from the Internet, the deployment procedure should not depend on Internet resources but rather envision that all required resources must be possible to be made available within the Local Automation Cloud.

Initially (after engineering but before deployment) there are the following systems and devices:

- A generic, configurable device, ready to be deployed
- A node in the Plant Description representing the device that is to be introduced.
- A node in the Plant Description for each system that a system on the device is intended to implement. Each of the system-nodes have at least one link to associate them with the device-node.

- Each planned service interaction between systems should be stored as service-links between the system-nodes in the Plant Description.
- A configuration in the Configuration Store associated with the device-node in the Plant Description. This configuration contains enough information to allow the device to host the desired systems and the NodeId by which each system-node is identified in the Plant Description.
- An identifier that is to be used during deployment to associate the device with the device-node identifier from the Plant Description. This may be achieved by using the NodeId from the Plant Description or by storing a specific identifier, such as a public key from a certificate, in the Configuration System beforehand.
- A user interface available during deployment should be able to navigate the Plant Description and transfer the previously mentioned identifier to or from the device through a local interface.

Deployment procedure:

1. Device is physically connected to the network and turned on.
2. Device connects to the network using DHCP, or a similar standardized technology applicable for the network in question.
3. Device looks for the Arrowhead core systems at predefined locations. (E.g. on the local network or a cloud service hosted by the device supplier)
4. Core systems authenticate device as factory configured device with basic authorization.
5. The user interface is now used to associate the physical device with the device-node in the Plant Description.
6. The device registers with mandatory core systems before access is granted to support core systems.
7. The device uses the associated identifier to access the Configuration store and retrieve its configuration.
8. The device configures itself to host systems according to the configuration, and the systems registers with the System Registry.
9. Depending on the implementation of Orchestration used, one of the following interactions is initiated:
 - (a) The System Registry accesses the Plant Description providing the association between the SystemId and the NodeId. The Plant Description system checks for all service-links to the associated NodeId and requests SystemId for all NodeIds connected by service-links. Using these provided SystemIds the Plant Description system creates orchestration rules for all affected systems.

- (b) The System Registry notifies the Orchestration system with the newly registered SystemId and associated NodeId. The Orchestration system then queries the Plant Description for system-nodes linked to the NodeId by service-links. It then uses the provided NodeIds to query the System Registry for associated SystemIds and creates the desired orchestration rules, possibly also using other available information such as QoS or similar.
- (c) The newly introduced system requests an orchestration from the Orchestration system, providing either SystemId or its NodeId. The Orchestration system in turn accesses the Plant Description (if necessary after getting the NodeId from the System Registry) which responds with the appropriate service-links. Using the service-links, and other available information, the Orchestration system provides the requested orchestration back to the newly introduced system.

At the end of the procedure, the operators and engineers should be informed if the procedure has been successful or if some part of it has failed. This can be done through an event or subscription mechanism, where involved user interfaces are notified depending on the result. For example, a deployment failure could first be sent to the user interface that was used in the deployment process, and escalated to other users only if the user that tried to deploy the device is unable to solve the issue.

5 Automation engineering: A process automation scenario

To visualise the engineering work-flow described in the previous section and to provide and basis for discussion in the following sections, this section presents a basic automation engineering scenario.

The engineering scenario is based on a water heating system, previously used as an example by Pang et al. [24]. In their work they show how an IEC 62424 Piping and Instrumentation Diagram (P&ID) can be formalised and used to generate automation application code using IEC 61499 function blocks. A P&ID for the scenario is presented in Figure 2, adapted from [24].

With regards to the engineering work-flow from idea to operation, as described in Section 1, this scenario assumes that there is a justified Conceptual application design, e.g., specifying that the three rooms F1R1, F2R1 and F3R1, should be heated using the district heating system that can be connected to in the Water supply room.

5.1 Functional design

This, first of the more technical phases of the engineering work-flow, consists of producing an overarching design. In most process automation engineering scenarios the P&ID constitutes a key design document, provided together with a few documents and lists of

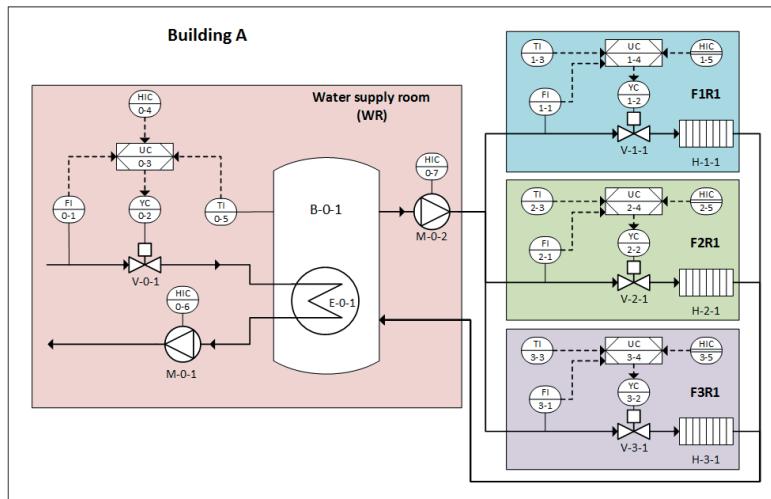


Figure 2: P&ID of a water heating system for a building with three rooms on different floors. Figure adapted from the design used by Pang et al. [24]

specifications, for detail engineering of the process and individual objects. A typical set of documents include:

- Piping and Instrumentation Diagram (P&ID)
- Functional design description
- Object list
- Technical requirements

In the case of this scenario, the documents

Piping and Instrumentation Diagram (P&ID)

The P&ID is the key design document for understanding the overall technical process and the standardised symbols used are well accepted in most applications where a material flow or between different systems is an important part of the design.

In the P&ID in Figure 2, there are two major process circuits. The external water supply enters the diagram from the left in the Water supply room, goes through valve V-0-1, into the heat exchanger E-0-1, and back through pump M-0-1 to the external supply network. This circuit effectively heats the water in the tank B-0-1 and is controlled through the controller UC-0-3. The second circuit circulates the water in tank B-0-1 through the pump M-0-2 into the three rooms where the radiators H-1-1, H-2-1 and H-3-1 heat one room respectively, before returning the water to B-0-1 for reheating. On

this circuit the temperature in each room is controlled by an independent control loop, by using control valves to restrict the flow through each radiator.

Functional design description

Accompanying the P&ID is usually a textual description of the functions, as a basis for programming and other engineering tasks. For the systems in Figure 2 there should be a description of what type of control should be implemented in each controller. As there are two sensor inputs (flow (FI) and temperature (TI)) connected to each controller and one actuator output (control valve (YC)), the functional design description should if the controller should be a cascaded PID control or if it should use a model predictive control algorithm. In short, it should provide the functional design aspects intended by the designer that was not captured in the P&ID.

Object list

The object list is a continuously updated document, describing all hardware components used in the system. In this scenario, the main objects would be all of the sensors, valves, pumps, radiators and user interfaces. The object list produced by the design team would include certain parameters to ensure the functionality, such as the control range of the valves and the required precision and accuracy of the sensors, while other parameters are to be decided by automation and electrical engineers.

In some cases the object list may be stored as a database, accessible by all design and engineering disciplines, but due to the diversity of engineering tools and data formats in use this is not the most common scenario.

For this scenario it is likely that the object list would include the tank and heat exchanger, the pumps, valves and radiators, and all of the sensors. Possibly, but not necessarily, the list may include controllers and human-machine interfaces (HMIs) as well.

Technical requirements

As a basis for further specification of parameters in the object list, a technical specification usually accompanies the design documents. This will include all of the implementation details that limits the options for electrical and automation engineers in their work. Often it will include performance requirements, non-functional requirements and references to applicable standards.

In this scenario, the technical requirements are likely to include environmental variables, such as required temperature sensor ranges and pump noise limits and size limits on radiators, as well as safety aspects and what kind of information should be available through different user interfaces. Additional information may be minimum flow through the valves that is required to keep the pumps within their operational limits.

5.2 Procurement and Engineering

As previously mentioned, much of the engineering consists of selecting appropriate components, assigning configurations, including operation parameters and program code or logic, and detailing all interactions between components.

Throughout the engineering process, engineers of different disciplines will be interacting with the object list, as the specification of some components may lead to requirements on other components.

Ideally, there are prepared design sections, or “typicals”, from a prepared library or from previous similar projects, that can help in the engineering. In the scenario of Figure 2, four very similar control loops can be identified consisting of a control valve (YC), one temperature sensor (TI) one flow sensor (FI), one user interface (HIC) and one controller (UC) each. These are likely to be prepared as two variations of a typical temperature control loop, one for the main circuit in the Water supply room and one for the three smaller circuits. The engineers may already have a set of sensors and valves that are known to work well together and for which there are electrical designs for each set and segments of control code that can be re-used.

In this scenario, the Procurement is most likely heavily influenced by the process performance for selecting process equipment (pumps, valves, sensors) and the availability of compatible “typicals” for selecting control equipment.

5.3 Deployment and Commissioning

As far as possible, all systems will be tested before delivery to the site, but as much of the functionality may depend on systems already at the site or on systems delivered by other suppliers, there are always limitations to how realistic the tests can be made.

At the site, all systems are connected according to the engineering documents and all configurations and control code is loaded into their respective devices. For some devices a configuration may be transferred electronically from generic configuration tools but many traditional devices require manual input of configuration parameters through specialised tools or manually with physical buttons or switches on the device.

As part of the commissioning, all systems are verified and tested rigorously, and all deviations must be documented either directly into the data store produced by the engineering, or in some other organised manner so that it can be included in the applicable documentation as soon as possible. This is vital, not only in order to have a valid set of documentation for the operational systems, but also to find possible improvements to the engineering work and typicals used.

6 Application of IoT Automation Engineering

The Engineering work-flow and deployment procedure described in Section 4 is expected to facilitate the engineering and deployment of IoT automation systems. This section is intended to illustrate the application of the work-flow, using the systems in Figure 2,

assuming a functional design providing the design documents as described in Section 5.1, and a library of service and system descriptions, as described by Blomstedt et al. [20].

For a qualitative evaluation of the engineering work-flow, the application is compared to the engineering of a traditional automation system based on a programmable logic controller (PLC) and Supervisor Control And Data Acquisition (SCADA) system. Significant similarities and differences are discussed for each part of the work-flow.

6.1 Functional design

For the scenario discussed here, much of the functional design is given through the P&ID. In its given form, the P&ID can be used to discuss if the components should be organised into one local automation cloud, or if there are advantages to keeping the four control loops in separate local clouds. In the original scenario there is no need for communication between the control loops, however, given the flexibility of an IoT automation system and the organised management of systems and system configurations included in the engineering work-flow, there are some interesting new options that could be added to this scenario.

For example, given the possibility to change the orchestration at run-time, if all of the components on Figure 2 belong to the same local cloud, it should be relatively easy to implement alternative control solutions and switch between operating modes for the whole building. Using the Arrowhead Framework, one such addition could be that the controller in the water supply room (UC-0-3) can be orchestrated to consume services directly from sensors or controllers in the different rooms, allowing for more efficient usage of the energy than what can be achieved through the control of a fixed temperature level for the tank. In such a re-engineering scenario, an engineer can prepare a new configuration of the UC-0-3 controller that incorporates the additional data in the control algorithm, and store it as an alternate configuration in the Configuration system. Additionally, appropriate service links are added to the Plant Description specifying the added service interactions. Once all additions are in place, a re-configuration of the UC-0-3 controller can be triggered, similar to steps 7-9 of the deployment procedure described in section 4.3, including the added orchestration rules generated from the plant description.

For the traditional engineering procedure, the adding of such connections would require re-configuration or re-programming of both the device providing the data and the PLC executing UC-0-3. In addition, a change between the two operating modes would either have to be programmed into the PLC beforehand, at a significant effort if this is not a standard solution, or the operation of the system would have to be halted as a new bit of code is downloaded into the PLC. Such a procedure may also incur additional commissioning effort.

Conclusively, the usage of a service-based solution, such as the Arrowhead Framework, is expected to clearly reduce the re-configuration effort to the system using the data, i.e. the controller UC-0-3 in this case.

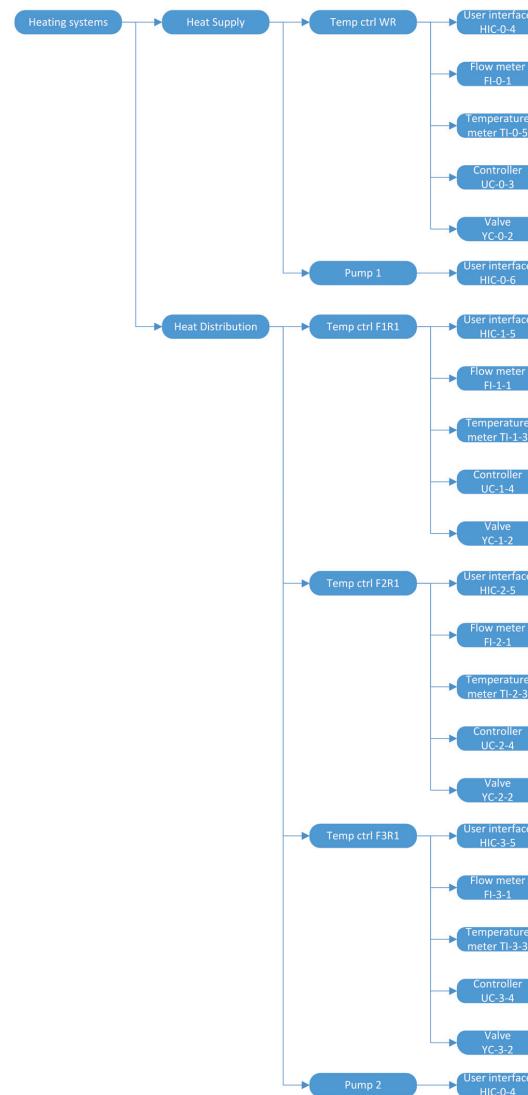


Figure 3: Plant Description of a functional hierarchy representing the control objects for the water heating scenario. The two main water circuits, Heat Supply and Heat Distribution, are identified as the main functional sections.

6.2 Procurement and Engineering

As described in Section 5.2, the procurement for this scenario is most likely dominated by the performance requirements on process equipment, but assuming that the IoT frame-

work in question has been widely adapted by device suppliers there should not be such a strong reliance on existing “typicals” for the selection of control hardware.

One of the anticipated benefits of an open IoT framework such as Arrowhead is that all suppliers are free to build compatible devices and systems that are interoperable with systems from other suppliers producing or consuming the same services. This is expected to open new possibilities in the procurement phase, as the “typicals” are no longer as important for easy combination of components.

As described in Section 4.2, the Engineering part of the IoT engineering work-flow depends to a large extent on the Arrowhead core systems Plant Description and Configuration. Their usage in this scenario is therefore highlighted in the following two subsections.

Utilising the Plant Description for engineering

As shown by Pang et al. [24], the P&ID can be formalised into objects that connected through different kinds of connections, this is essentially what the Plant Description is intended to abstract from the engineering data. One possible abstraction from the P&ID in Figure 2, is shown in Figure 3. It should be noted that this illustration only contains the objects and a functional hierarchy for navigation, as if at a state when additional links in the Plant Description, representing e.g. the signal interfaces between the objects, have not yet been added.

At this point the Plant description provides an overview of the objects identified in the P&ID, as illustrated in Figure 3, and each of these objects can be given an identifier in the data set for IoT devices selected by procurement, allowing identification and synchronisation between engineering disciplines.

As the engineers work with other parts of the design data, further relations between objects may be identified and added to the Plant Description, automatically if the source data is digital and following a compatible standard, or otherwise manually.

Assuming that each object in the P&ID is implemented by a separate software system, the signal connections in the P&ID can directly be interpreted into desired service interactions and stored in the Plant Description as *Service Links*, for future Orchestration of the systems. For more complex scenarios with multiple possible service interactions between systems, there is a need for a more advanced user interface, either to the Plant Description or to have service interactions captured in the existing design and engineering tools from which the Plant Description abstracts its data.

As all Systems and Service Links are identified and stored in the Plant Description, they can be used to generate Orchestration rules, as discussed in [14], and illustrated with Figure 4.

Usage of Configuration Store during engineering

The Configuration Store should be used as a repository for configuration files during the engineering phase, managed using the object identities from the Plant Description. If the

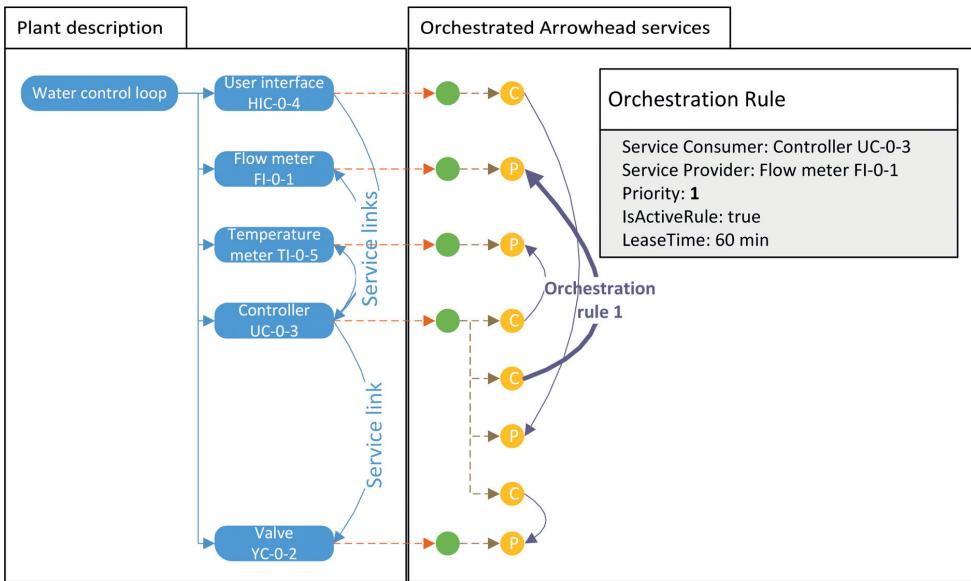


Figure 4: Orchestration rules constructed from the Service Links in the Plant Description. The green circles indicate active Arrowhead systems and the yellow circles indicate services that are produced (P) or consumed (C) by the systems.

configuration files are in a clear text format, the configuration files can be directly compiled from engineering data, allowing for quick updates through non-proprietary tools.

As discussed in [14], the benefits of a Configuration Store are larger in the deployment phase and at later stages during the operation and maintenance where upgrades and replacements may be an issue if configurations are poorly managed. However, even for a small installation the systematic approach to device identities and the automated configuration approach means that the testing is easier and can better predict the situation during installation at the site. As IoT-devices can be expected to be more configurable than their traditional counterparts, where e.g. a temperature sensor operates more like a thermistor, there may be more configurations to manage in the IoT-solution than in the PLC-case.

Qualitative comparison with traditional engineering methods

The engineering effort for the traditional SCADA is most likely similar to the engineering of a SOA-based user interface and supervisory control system. At least on a qualitative level it is difficult to identify any significant differences between the two.

For this scenario, a PLC-based system is likely to consist of one central PLC executing all of the control loops and communicating to I/O-nodes in each room through a fieldbus network. Each control loop could be programmed using the prepared “typicals”,

with minor adjustments. As mentioned in Section 5.2, this scenario is likely to have typicals available for each control loop, meaning that the engineering is mostly limited to assigning the specific performance and communication parameters associated with each component. The extent of the effort depends mostly on the hardware and field-bus technology selected.

If there are “typicals” available, the engineering work for such a small scenario as this is fairly similar for the IoT automation systems and for the PLC-based solution. The effort of modifying the typicals and for configuring IoT systems is qualitatively at the same level.

However, one significant difference appears if the systems are to be tested off-site before mounting them at the site, as the traditional technology requires individual configuration. This means that it is very important for the traditional approach that the devices are clearly identified so that the installation at the site can be made exactly as the tested set-up. Otherwise additional configuration has to be made at the site, possibly under pressure of time and at less comfortable working conditions.

This means that even though the configuration effort may be similar in this phase, the Configuration store is expected to benefit the deployment phase, as well as later on during operation and maintenance where life-cycle management of device configurations may become an issue.

6.3 Deployment and commissioning

At the time of deployment, all devices are delivered to the site where they are to be installed. For the proposed engineering work-flow, a significant advantage to traditional solutions is that, as long as all components are compliant with the procedure, they can be delivered directly to the site from the system supplier, without any special configuration on the hardware.

Following the physical connection of a device or a small group of devices, the technician uses his trusted mobile device to extract a device identity from the newly connected device and associate it with the logical position or functionality described in the engineering documents or Plant description (enumerated as step 5 in the Deployment procedure, Section 4.3). E.g. for this specific scenario, the technician connecting a temperature sensor in the first room of the first floor (F1R1) should identify that sensor as “TI-1-3”, and so on. The sensor can subsequently follow the rest of the procedure and ask the Configuration Store for its configuration and the Orchestration system for its orchestration rules, allowing it to connect and start operating as designed.

If the Plant description contains more than one hierarchy for system management, as is supported by e.g. IEC 81346, the technician may be able to navigate a structure based on the location of devices, rather than functionality. If so, there would be one sub-tree of the hierarchy labeled “F1R1”, within which he could find all of the devices to be installed in that room. This could help in the part of this scenario where the pump M-0-2 is to be installed, as this object alone belongs to the function “Heat Distribution” but is located in the Water supply room.

Qualitative comparison with traditional engineering methods

In the case of deployment of a traditional PLC-based system, it is likely that the technician is limited to a binder of paper drawings and lists for both installation and documentation of the installation work, as all installed components and wires are usually marked manually directly on the a copy of the paper drawings, first once as they are installed and once more in a different colour when they the connections are electrically verified.

As the IoT devices are expected to use wireless or Ethernet connections, capable to automatically verify the connectivity over the connected network, at least the verification is expected to be easier. If the connections are wireless, the installation can be significantly cheaper.

Given more capable devices, able to configure themselves as part of the deployment procedure, the commissioning is also expected to be significantly more efficient using the IoT engineering approach, as less effort is spent on verifying individual signals the commissioning can more quickly start to verify system functionality, meaning faster commissioning as well.

7 Conclusion

Given the qualitative evaluation throughout Section 6, it is clear that there are some significant advantages to the presented engineering work-flow, compared to traditional automation engineering procedures. However, it has to be pointed out that there are also other methods to improve the traditional engineering procedures that have not yet been generally accepted, as was mentioned in Section 2, making the comparison less straight-forward.

In general, it can be argued that the analysis indicates that the presented engineering work-flow appears to promote and support the expected flexibility of IoT-based automation systems, through systematic management of service interactions and system configurations. The analysis further indicates that an IoT-automation engineering work-flow, based on the Arrowhead Framework, significantly decreases the effort for certain phases of the engineering work, without significant increases in any other phases.

As discussed briefly regarding the Functional design in Section 6, IoT automation systems are expected to enable or simplify functionality that is not traditionally used. Such as new functionality may require modifications to existing tools and standards for design and engineering. A significant challenge in this area will be how to merge these new possibilities with existing procedures, in a way that allows cooperation between the experienced engineers and designers used to traditional tools and the technical experts with knowledge of the possibilities provided by the new technology.

8 Future work

For a more empirical, quantitative, evaluation of the engineering work-flow certain concepts would need to be detailed and implemented. Chief among these would be a more

complete implementation of the Plant description, able to utilise raw engineering data from existing standardised tool and produce functioning orchestration rules.

A different route to a more detailed evaluation would be further studies of automation systems in other domains where IoT automation systems are expected to be deployed, preferably systems where a real implementation of other solutions have already been made, so that there is a clear baseline of the engineering effort for those solutions.

The work-flow described is currently focused on the engineering of a production process, but given that data is exchanged regularly between engineering tools it should be possible to define similar interaction for the engineering tools used in product design. Such interaction would increase the usefulness of the approach as it could possibly enable automatic re-configuration of production lines, based on the engineering data of new products. This would extend the work-flow from the traditional engineering phases into operation of the systems.

Furthermore, the management of local automation clouds should be further discussed and evaluated. In this area there are interesting aspects in how to select the size of a local cloud, and what systems are necessary and desirable within each local cloud. These aspects should be reviewed from a broad set of perspectives, including security, performance, efficiency, manageability, and others.

Acknowledgment

The authors would like to extend their gratitude towards EU ARTEMIS JU for funding within project ARTEMIS/0001/2012, JU grant nr. 332987 (ARROWHEAD). We would also like to thank the partners of the Arrowhead project for fruitful collaboration and interesting discussions.

References

- [1] J. Lee, B. Bagheri, and H.-A. Kao, “A cyber-physical systems architecture for industry 4.0-based manufacturing systems,” *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.
- [2] B. Scholten, *The Road to Integration: A Guide to Applying the ISA-95 Standard in Manufacturing*. ISA, 2007.
- [3] Arrowhead project. [Online]. Available: <http://www.arrowhead.eu>
- [4] *ISA95, Enterprise-Control System Integration*, ISA Std.
- [5] J. Delsing, Ed., *IoT Automation - Arrowhead Framework*. CRC PRes, To appear Feb 2017 2016.
- [6] R. Yang, *Process Plant Lifecycle Information Management*. AuthorHouse, 2009. [Online]. Available: <https://books.google.se/books?id=zmgzdzf1GR0C>

- [7] I. Scheeren and C. E. Pereira, "Combining model-based systems engineering, simulation and domain engineering in the development of industrial automation systems: Industrial case study," in *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, June 2014, pp. 40–47.
- [8] P. Logan, D. Harvey, and D. Spencer, "Documents are an essential part of model based systems engineering," *INCOSE International Symposium*, vol. 22, no. 1, pp. 1899–1913, jul 2012. [Online]. Available: <http://dx.doi.org/10.1002/j.2334-5837.2012.tb01445.x>
- [9] T. Moser, S. Biffl, W. D. Sunindyo, and D. Winkler, "Integrating production automation expert knowledge across engineering stakeholder domains," in *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*, Feb 2010, pp. 352–359.
- [10] T. Moser and S. Biffl, "Semantic integration of software and systems engineering environments," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 1, pp. 38–50, Jan 2012.
- [11] S. Biffl and A. Schatten, "A platform for service-oriented integration of software engineering environments," in *Proceedings of the 2009 Conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the Eighth SoMeT 09*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2009, pp. 75–92. [Online]. Available: [{http://dl.acm.org/citation.cfm?id=1659308.1659316}](http://dl.acm.org/citation.cfm?id=1659308.1659316)
- [12] S. Biffl, A. Schatten, and A. Zoitl, "Integration of heterogeneous engineering environments for the automation systems lifecycle," in *2009 7th IEEE International Conference on Industrial Informatics*, June 2009, pp. 576–581.
- [13] O. Carlsson, D. Vera, J. Delsing, B. Ahmad, and R. Harrison, "Plant descriptions for engineering tool interoperability," in *2016 14th IEEE International Conference on Industrial Informatics (INDIN)*, 2016.
- [14] O. Carlsson, C. Hegedűs, J. Delsing, and P. Varga, "Organizing IoT Systems-of-Systems from Standardized Engineering Data," in *Industrial Electronics Society, IECON 2016 - 42nd Annual Conference of the IEEE*, 2016.
- [15] O. Carlsson, P. P. Pereira, J. Eliasson, J. Delsing, B. Ahmad, R. Harrison, and O. Jansson, "Configuration service in cloud based automation systems," in *Industrial Electronics Society, IECON 2016 - 42nd Annual Conference of the IEEE*, 2016.
- [16] P. Varga, F. Blomstedt, L. L. Ferreira, J. Eliasson, M. Johansson, J. Delsing, and I. M. de Soria, "Making system of systems interoperable – the core components of the arrowhead framework," *Journal of Network and Computer Applications*, aug 2016. [Online]. Available: <http://dx.doi.org/10.1016/j.jnca.2016.08.028>

- [17] J. Delsing, J. Eliasson, J. van Deventer, H. Derhamy, and P. Varga, "Enabling IoT automation using local clouds," in *Proceedings World Forum - IoT 2016*. IEEE, Dec. 2016.
- [18] C. Hegedűs, D. Kozma, G. Soós, and P. Varga, "Enhancements of the Arrowhead Framework to refine inter-cloud service interactions," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 5259–5264.
- [19] L. L. Ferreira, M. Albano, and J. Delsing, "QoS-as-a-Service in the local cloud," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sept 2016, pp. 1–8.
- [20] F. Blomstedt, L. L. Ferreira, M. Klisics, C. Chrysoulas, I. M. de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, "The arrowhead approach for soa application development and documentation," in *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, Oct 2014, pp. 2631–2637.
- [21] D. Schulz, "FDI and the Industrial Internet of Things," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015, pp. 1–8.
- [22] G. Lilis, G. Conus, N. Asadi, and M. Kayal, "Towards the next generation of intelligent building: An assessment study of current automation and future iot based systems with a proposal for transitional design," *Sustainable Cities and Society*, vol. 28, pp. 473 – 481, 2017. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S2210670716302414>
- [23] C. Li, N. Wang, and J. Zhu, "The Unified Management Platform of IEC61850 Configuration Files Based Plug-In," *Energy Procedia*, vol. 17, Part B, pp. 1441 – 1446, 2012, 2012 International Conference on Future Electrical Power and Energy System. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1876610212006005>
- [24] C. Pang, V. Vyatkin, and W. Dai, "IEC 61499 based model-driven process control engineering," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Sept 2014, pp. 1–8.

PAPER H

The Arrowhead Framework architecture

Authors:

Jerker Delsing, Pal Varga, Luis Ferreira, Michele Albano, Pablo Puñal Pereira, Jens Eliasson, Oscar Carlsson and Hasan Derhamy

Reformatted version of paper originally published in:

Book chapter, IoT Automation: Arrowhead Framework, Taylor & Francis, 2017.

© Taylor & Francis 2017 From IoT Automation: Arrowhead Framework by Oscar Carlsson, author Jerker Delsing, editor. Reproduced by permission of Taylor and Francis Group, LLC, a division of Informa plc.

Chapter 3 - The Arrowhead Framework architecture, 2017, Jerker Delsing, Pal Varga, Luis Ferreira, Michele Albano, Pablo Puñal Pereira, Jens Eliasson and Hasan Derhamy, *The Arrowhead Framework architecture*, Taylor & Francis, 2017.

1 Architecture fundamentals

The objective of the Arrowhead Framework architecture is to facilitate the creation of local automation clouds. Thus enabling local real time performance and security, paired with simple and cheap engineering, while simultaneously enabling scalability through multi-cloud interaction.

The architecture addresses the move from large monolithic organisations towards multi-stakeholder cooperation where cooperation is fostered by market requirements. This is to support the high-level topics in today's society such as sustainability, flexibility, efficiency, and competitiveness in production.

Devices in such local clouds are considered to be IoT devices speaking at least one SOA (Service-Oriented Architecture) protocol. The capability of building automation systems requires a number of local cloud properties to be enabled. Furthermore, both intra- and inter-cloud information service exchange capabilities are necessary for enabling IoT devices to inter-operate and to be integrated with others to become an automation System of Systems. Previous work in this field comes from several larger EU projects such as Socrates and IMC-AESOP. The Arrowhead Framework architecture is building on the results of these projects [1, 2].

To facilitate this objective, the Service-Oriented Architecture paradigm is used. Thus the following properties are important starting points:

- Loose coupling
 - Autonomy - a service exchange is not supervised
 - Distributed - services are distributed over several devices
 - A system is responsible, owns the information, and can decide whom to share with
- Late binding
 - Possible to use information any time by connecting to the correct resource at a given time
- Lookup
 - Publish and register services to notify others about endpoints (how to reach me)
 - Discover others that I comply with (expected/wanted ServiceType)

The design of the Arrowhead Framework is further based on the following fundamentals:

- A system producing a service has the initial authority of its own service offering
- Information assurance shall be at the service exchange level

- Information centric networking

The Arrowhead Framework allows for

- A Publish-Subscribe approach
- Both the Push and Pull approaches
- Dynamic creation of new services and their subsequent usage

The above properties, fundamentals, and functionalities are provided by the Arrowhead Framework through

- A minimal set of mandatory services to create a System of Systems
- A set of automation support services - facilitating design of application System of Systems

A developer needs to know how to develop, deploy, maintain, and manage Arrowhead compliant systems. Therefore, it is crucial that there is a common understanding of how Arrowhead Framework services, systems, and System of Systems are defined and described. To address these issues, the framework also includes design patterns, documentation templates and guidelines that aim at helping systems, newly developed or legacy, to conform to Arrowhead Framework specifications.

In the following we will discuss the Arrowhead Framework local automation cloud architecture. The architecture and a number of the core services implementing the architecture are open source. The Arrowhead Framework community development site is located at http://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page also reachable from <http://www.arrowhead.eu>.

2 Important definitions

To discuss and define a local cloud architecture, a few definitions are important. The Arrowhead Framework local automation cloud architecture makes use of the following important key words:

- Service
- System
- Device
- Local cloud
- System of Systems

These key words may have other definitions in different domains and contexts. But, in this book and within the Arrowhead Framework the following definitions are used.

2.1 Service

In the context of the Arrowhead Framework, a service is what used to exchange information from a providing system to a consuming system, see Figure 1. In a service, capabilities are grouped together if they share the same context [3]. A service can be implemented to use a number of different SOA protocols. Some examples of SOA protocols are REST [4], COAP [5], XMPP [6], MQTT [7, 8], or OPC-UA [9].

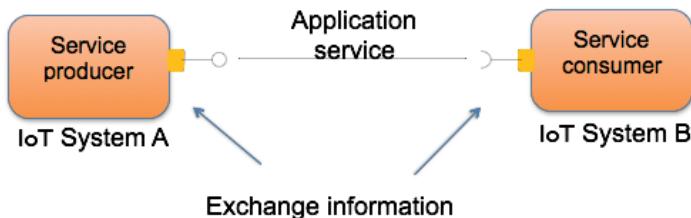


Figure 1: Services provide information exchange between a service-producing system and a service-consuming system.

A service is produced by a software system (see below). A service can have associated meta-data and can be capable of supporting non-functional requirements such as security, real-time operation, or different levels of reliability among others.

It must be possible for an Arrowhead Framework compliant service to

- be registered with the Arrowhead Framework mandatory core systems
- be consumed or provided by an Arrowhead Framework compliant system

A service may be capable of

- being dynamically configured

2.2 System

An Arrowhead Framework system is what is providing and/or consuming services, see Figure 2. A system can be the service provider of one or more services and at the same time the service consumer of one or more services. A system is implemented in software and executed on a device (see below). A system can have associated meta-data. We are here separating the software based system from the hardware - device. The reason is grounded in security considerations. To achieve a chain of trust, a piece of “computing” hardware need to be identifiable. The same is required for a software application executing on that hardware. In the Arrowhead context an software application capable of producing and/or consuming services is named system and the “computing” hardware hosting a system is named device.

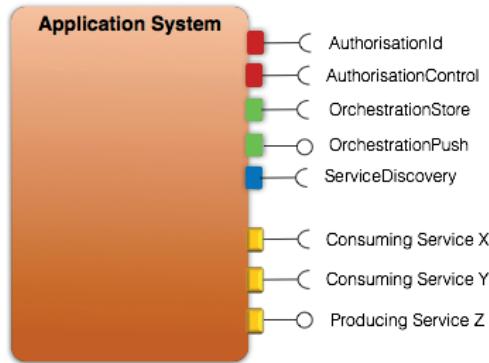


Figure 2: A system is capable of consuming the Arrowhead Framework mandatory core services and will produce and/or consume one or more services.

An Arrowhead Framework compliant system shall be capable of:

- consuming Arrowhead Framework compliant services
- producing Arrowhead Framework compliant services
- registering itself to the mandatory SystemRegistry
- releasing its authority of its own service offering to a local cloud orchestration system

A system may be capable of

- creating new services on demand
- being dynamically configured

2.3 Device

An Arrowhead compliant device is a piece of equipment, machine, hardware, etc. with computational, memory and communication capabilities which hosts one or several Arrowhead Framework systems and can be bootstrapped in an Arrowhead Framework local cloud, cf. Figure 3. Any other device, equipment, machine, hardware, component etc. is non-Arrowhead compliant.

A device may be capable of

- dynamically hosting new systems and their services
- being registered to the DeviceRegistry
- being dynamically configured

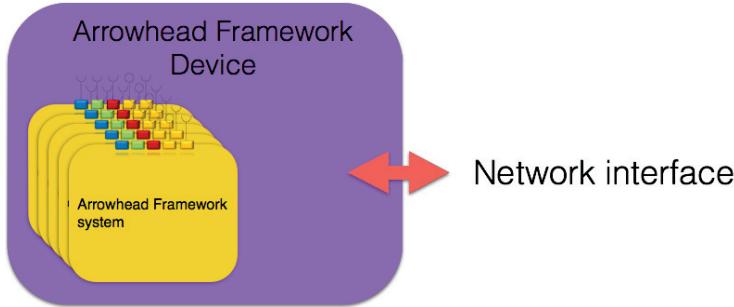


Figure 3: An Arrowhead Framework device is a equipment capable of hosting systems exchanging services.

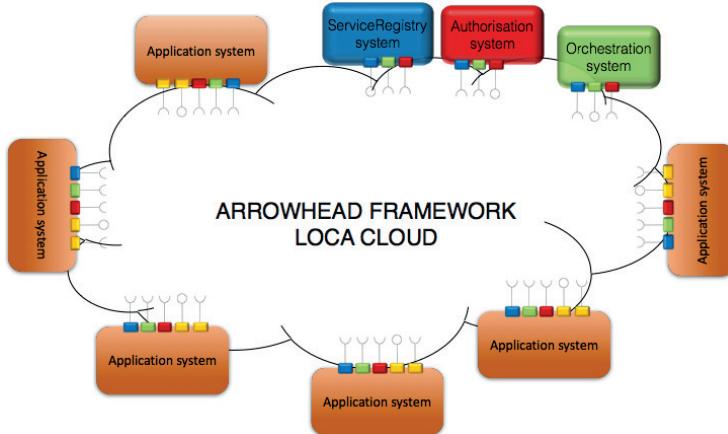


Figure 4: An Arrowhead Framework local cloud hosts at a minimum the three mandatory core services, ServiceRegistry, Orchestration, and Authorisation and one or more application systems.

2.4 Local cloud

In the Arrowhead Framework context a local cloud is defined as a self-contained network with the three mandatory core systems deployed and at least one application system deployed, cf. Figure 4. A local cloud shall host only one ServiceRegistry system. For administrative and security reasons it is strongly recommended that only one instance of the other two mandatory core services are deployed in a local cloud.

It is advisable that a local cloud holds a mean of distributing IP addresses to joining devices. It is further advisable that the local cloud has firewall protection to surrounding networks. By which external network traffic can be blocked from reaching the interior of the local cloud.

2.5 System of Systems

A System of Systems within the Arrowhead Framework is defined as a set of systems, which are administrated by the Arrowhead mandatory core systems and exchange information by means of services.

A local cloud thus becomes a System of Systems in the Arrowhead Framework's definition. If two systems reside in different local clouds that are administrated by Arrowhead core systems to exchange services, it also is a System of Systems in the Arrowhead Framework's definition.

When Arrowhead compliant systems collaborate, they become a System of Systems. Since two or more such Systems of Systems can collaborate, the Arrowhead Framework becomes a natural enabler of further, complex solutions. Figure 5 depicts such an example.

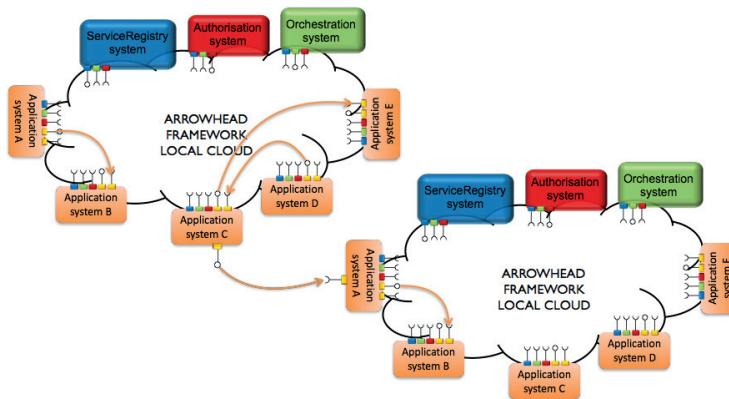


Figure 5: Arrowhead Framework Systems exchanging services, thus building a system of systems within a local cloud and between local clouds.

Service exchanges between systems can be initiated by an orchestrating system providing orchestration rules to the involved systems. Service exchanges can also be initiated by apriori knowledge of one system to seek a specific service via a ServiceRegistry. Upon a seek criteria match a service exchange can be initiated. To support a structured governance of a System of Systems, the preferred approach is the use of an Orchestration system.

2.6 Service, system, device and local cloud identifiers

In order to allow service discovery, system administration, and device mappings, there is a need to uniquely define identifiers to the devices, systems, and services. Thus, to enable proper identification of which device and system a specific service is executed by,

there is a need for a way of identifying

- Services
- Systems
- Devices
- Local cloud

Here references are made to the SysD, SD, IDD, CP and SP documents. For definitions of these documents please see Section 3 below.

There are two classes of Arrowhead Framework services:

- Core services, sub-divided into
 - Mandatory core services
 - Automation support services
- Application services

The identification of Arrowhead Framework services follows the same idea as identifiers in DNS-SD records [10, 11, 12]. Thus an Arrowhead Framework service is identified using the following structure, which follows the RFC-6335 recommendations [13]:

- Core services
 $_ServiceName._ahfc-ServiceType._protocol._transport.domain$
- Application services
 $_ServiceName._ahf-ServiceType._protocol._transport.domain$
- $.domain$ is the domain name under which the device with an associated system has an IP address, e.g., $subdomain.domain.topdomain$. An example of a domain name can be $app.arrowhead.eu$.

The detailed explanation is

- $_ServiceNames$ is the name of the particular service instance, e.g., $_Temp102$.
- $_ahf/c\text{-}servicetype$: The Arrowhead Framework makes use of the selective service instance enumeration (subtypes) possibility as described in RFC-6763 [10]. For core services the ServiceType always starts with $_ahfc$ - and $_ahf$ - is reserved for Arrowhead Framework application services.
 - $_ahfc\text{-}ServiceType$ is an Arrowhead Framework core services type where e.g. the service type orchestration is added, leading to the complete ServiceType of $_ahfc\text{-}orchestration$. The above specification should be given in the SD (Service Description) document.

- *_ahf-ServieType* is an Arrowhead Framework application services type. Here e.g. vibration is the service type, leading to the complete ServiceType of *_ahf-vibration*. Above specification should be given in the SD document.
- *_protocol* is, e.g., *_coap* when the CoAP [5] protocol is used and specified in the IDD (Interface Design Description) document.
- *_transport* is, e.g., *_udp* when the UDP [14, 15] transport protocol is used and specified in the IDD document.

The end point to an application service instance consists of a path and a port. Using, for example, REST [4], an end point may look like <http://app.arrowhead.eu/ahf/Temp1/8090/>.

To enable discovery of interoperability at the service level, there is a need for information on payload encoding, compression and semantics. In the context of the Arrowhead Framework, this is regarded as service meta-data. Service meta-data is to be provided through the DNS TXT record using the following key pairs:

- *encode=syntax*, e.g., *encode=xml* when XML [16] encoding is used and specified in the CP (Communication Profile) document.
- *compress=algorithm*, e.g., *compress=exi* when EXI [17] compression is used and specified in the CP document
- *semantic=XX*, e.g., *semantic=senml* when SenML [18] semantics is used and specified in the SP (Semantic Profile) document.

A device is identified through a DeviceName plus the associated MAC address of its network interface. Hereto an ID key shall/may be associated using a secure bootstrap process; see Section 4.4. This enables us to associate the device hardware and its network interface to its IP address and MAC address, allowing for building service exchange topology information useful for quality of service management. For a device with more than one network interface device instances with the same DeviceName but different MAC addresses should be defined. The above information shall be specified in the SysD (System Description) document.

An Arrowhead Framework software system is identified through a system ID key generated from a two-way asynchronous authorisation process involving a trusted part and the device name instance identifying which device hardware is hosting the system and which network interface the software system is using. This shall be specified in the SysD document.

A local cloud is identified with the servicename and service type of the ServiceRegistry system. An example is *_cloud-x._ahfc-ServiceDiscovery*.

The following four service types are considered to be well known:

- *_ahfc-ServiceDiscovery*
- *_ahfc-SystemDiscovery*

- *_ahfc-DeviceDiscovery*
- *_ahfc-AuthorisationControl*

This allows for simple deployment with authorisation registration of devices, systems and services with the local cloud.

3 Documentation structure

It is a common experience of both system developers and integrators that insufficient and not properly structured documentation makes it hard – if not impossible – to properly understand how to integrate with a given system.

A system integrator needs to know how to develop, deploy, maintain, and manage Arrowhead compliant systems. To address these issues, the Arrowhead Framework document structure allows documenting SOA artefacts in a common format [19].

For this purpose the Arrowhead consortium [20] defined a three-level documentation structure: System of Systems, system, and service level. These are depicted in Figure 6, which also shows the links between documents.

The main concept of the documentation structure is to provide abstract and implementation views of the Systems of Systems, systems, and services. The main purpose of the abstract view documents is to allow any developer to implement System of Systems, systems and services, based on these documents. This further support that the resulting implementation becomes Arrowhead Framework compliant.

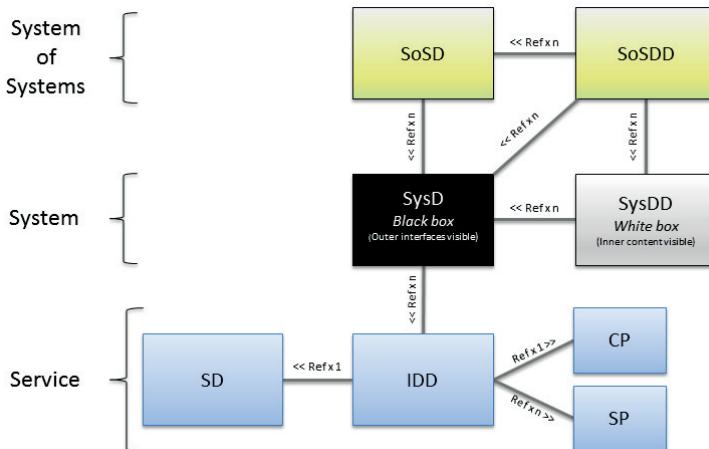


Figure 6: Arrowhead Framework documentation relationships.

3.1 System of Systems level documentation

The System of Systems level consists of two types of documents. The System of Systems Description (SoSD), which shows an abstract view of an SoS, and the System of Systems Design Description (SoSDD), which shows an implementation view of the SoS with its technologies and deployment views.

The SoSD describes the main functionalities and the generic architecture of the SoS. It will mainly be used to describe one System of Systems in an abstract way, without instantiating into any specific technologies. Examples of its usage are the description of generic SOA-based installations, like building automation systems or a factory automation system. The document should present its main building blocks as independent systems with pointers to their specific abstract view documents, the system Descriptions (SysDs) see below. Also, diagrams representing system behaviour, like use-case diagrams and behaviour diagrams (e.g., using UML, BPMN, SysML, AutomationML [21, 22, 23, 24]) must be included. This document also includes information about non-functional requirements, like required levels of QoS and security.

For the Arrowhead Framework itself there is a generic SoSD for which the current version can be found at the Arrowhead Framework wiki [25].

The SoSDD document describes how an SoSD has been implemented on a specific scenario, showing the technologies used and its setup. Therefore, it points out all necessary black box SysD and white box SysDD documents, describing the systems used in this realisation. The SoSDD should also contain behaviour diagrams which clearly identify the technologies used and the setup of this SoS realisation. The document can optionally include a description of its physical implementation and the non-functional requirements implemented by this realisation.

3.2 System level documentation

System level documentation consists of a black box System Description (SysD) document and a white box System Design Description (SysDD) document.

The SysD describes the system as a black box, documenting the system functionality and its hosted services and their provided and required interfaces with the corresponding technical solutions, without describing its internal implementation. The by the system provided service interfaces are referenced and defined in the Interface Design Description (IDD) document; see Section 3.3 below. The services provided are defined in the Service Design (SD) document, see Section 3.3 below. The SD document shall provide a clear definition of how to interface the system, thus enabling coding of a consumer system.

The SysDD extends the black box description, showing its internal details. This document is optional, since it might expose knowledge of the company which implemented the system, but it can be used as an internal document for future reference by the system owner.

3.3 Service level documentation

Service level documentation consists of four documents: the Service Description (SD), the Interface Design Description (IDD), the Communication Profile (CP), and the Semantic Profile (SP).

The IDD is pointed to by a SysD document. It states the actual implemented solution of a system. Here are defined the service identifiers of the specific service implementations. For the SOA protocol and encoding used the IDD is making reference to the Communication Profile (CP) document, see below. For data and information semantics the IDD make reference to the Semantics Profile (SP) document; see below.

The SD is a technology independent and abstract view of a service. The document describes the main objectives and functionalities of the service and its abstract interfaces. Further, an abstract information model shall be provided. Sequence Diagrams showing how the service is interacted with, shall also be provided.

The CP contains all the information regarding the transfer protocol, data compression, data encryption, and data encoding used, e.g., CoAP, UDP, EXI, DTLS, and XML.

The SP defines the data and information semantics used, e.g., SenML.

4 Arrowhead Framework architecture

Based on the SOA fundamentals and principles discussed previously, a local cloud will require three fundamental properties:

- Capability to register a service to the local cloud
- To discover which services are registered with the local cloud
- Enabling loosely coupled data exchange between producer and consumer systems
 - orchestrate service exchanges
- Authentication of consuming systems and granting Authorisation of service exchanges

Following the above fundamentals, definitions, and documentation structures, a local automation cloud architecture is defined. The architecture is composed of a number of systems, which provide a number of services. The objective is an architecture from which self-contained local automation clouds can be created. These clouds shall further be capable of providing certain automation support services, and provide support for bootstrapping, security, suitable metadata, protocol and semantics transparency, and inter-cloud service exchanges.

For that purpose the architecture features three types of services:

- Mandatory core services
- Automation support core services

- Application services

These services are provided by mandatory and support core systems, as well as application systems.

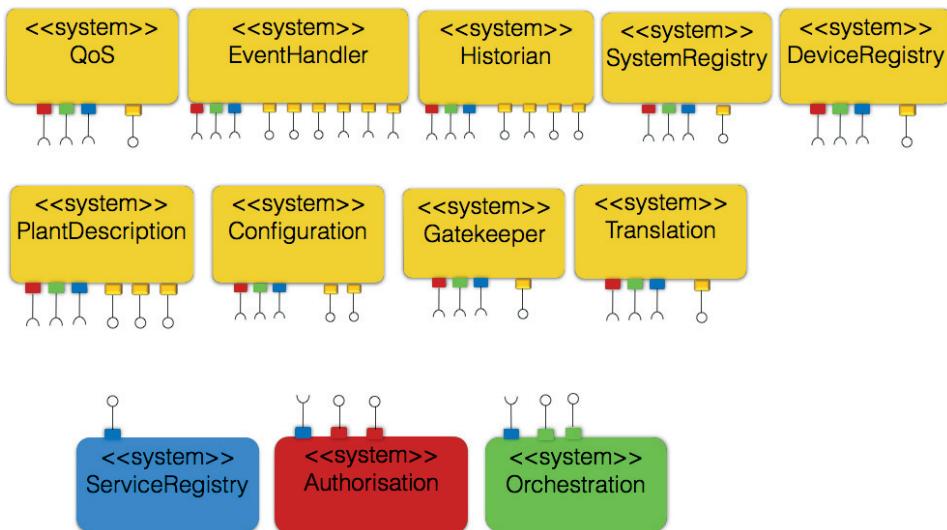


Figure 7: Core systems of the Arrowhead framework.

The use of this set of mandatory core systems and services makes it possible to design and implement a minimal local automation cloud. These mandatory core services will enable the desired basic properties of a local cloud, as discussed in Chapter 2. In order to support important local cloud properties, a number of automation support core services are defined as part of the architecture. These core services are provided by core systems. Figure 7 summarises the core systems currently defined with the Arrowhead Framework. Utilising all of these core systems in a local cloud is not mandatory. Implementations and documentation of the mandatory core systems and many of the automation support core systems are available as open sources via the Arrowhead Framework wiki [25].

In the following sections the core systems providing core services are described at a system level, corresponding to the SysD document of the Arrowhead Framework documentation structure.

4.1 The mandatory core systems

These mandatory core services support the following fundamental properties assigned to a local cloud:

- Loose coupling

- Autonomy - a service exchange is not supervised
- Distributed - systems exchange services are distributed on several devices.
- A system is responsible and owns its information and decides with whom to share
- Late binding
 - Possible to use information any time by connecting to the correct resource at a given time
- Lookup
 - Publish and register to notify others about endpoint (how to reach me)
 - Discover others that I comply with (expected/wanted service type)
- Information assurance at service exchange level
- Minimal set of mandatory services to create a System of Systems

The mandatory core systems and their service provide these fundamental properties to a local cloud. Thus enabling and allowing service exchanges between a service producer and a service consumer with desired level of security and autonomy, which is briefly illustrated in Figure 8 .

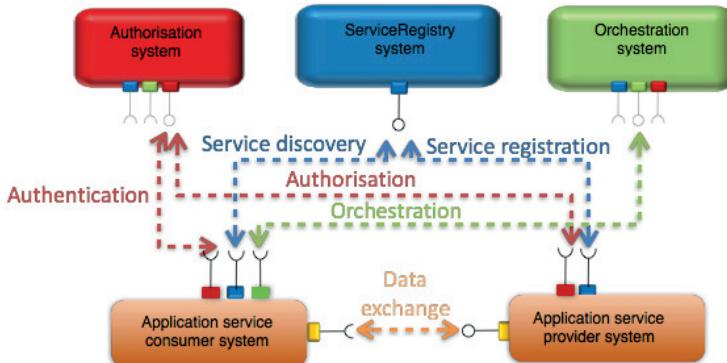


Figure 8: Minimal local cloud with indications of mandatory core service interaction enabling service exchange between two application systems.

Therefore, the creation of a minimal working local cloud based on the Arrowhead Framework must be based on three mandatory core services. The mandatory core services and their three hosting systems are

- ServiceRegistry system providing the

- ServiceDiscovery service
- Authorization system providing the
 - AuthorisationControl service
 - AuthorisationManagement service
 - AuthenticationID service (only for ticket-based implementation)
- Orchestration system providing the
 - Orchestration service
 - OrchestrationManagement service

The objectives of these mandatory core systems with their system definitions and high-level descriptions are given below. In Chapter I the Arrowhead Framework implementation of the core systems and services is provided in some detail. For the current status of these implementations, please refer to the Arrowhead Framework wiki [25].

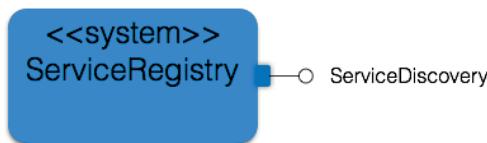


Figure 9: The mandatory ServiceRegistry system.

ServiceRegistry system (mandatory)

The objective of the ServiceRegistry system is to provide storage of all active services registered within a local cloud and enable the discovery of them.

The ServiceRegistry system keeps track of all active services produced within a local cloud. It provides a service registry functionality based on DNS and DNS-SD standards [10, 11, 12], since the Arrowhead Framework is a domain-based infrastructure. The ServiceRegistry is an independent system that provides one service and does not consume any other services, cf. Figure 9. All ServiceRegistry system graphs are color coded in blue.

All systems within the local cloud with services producing information to the local cloud shall publish their service within the ServiceRegistry by using the ServiceDiscovery service. Using the DNS-SD engine, in the ServiceRegistry system, services can be published, un-published or looked up. The ServiceRegistry system holds all published and active services within the local cloud. A cleaning mechanism for broken services is still to be defined. The usage of the DNS-SD for storing registered services is the basis for the service identifier structure described in Section 2.6.

All systems within the local cloud with services that produces information shall publish their producing service with the ServiceRegistry system by using the ServiceDiscovery service. If a system disconnects from the local cloud, its services shall be de-registered in the ServiceRegistry system. To address broken systems and network failures a registration time out and keep alive approach is recommended. The DNS PTR and DNS SRV, TimeToLive, TTL, record provide such time out data. Thus a system should re-register its services according to this TTL data. If not done the ServiceRegistry can unpublish the service.

The details of a service layer agreement, SLA, holds information on service protocol, transport protocol, interface, associated methods, datatypes, encoding, semantics, and compression. There are several approaches to provide this information. The obvious one is from design time documentation. The more attractive approach is to provide such information as meta-data with the registered service. In this way the SLA provided by the service producer can be retrieved and decoded by a consumer.

There exist some technologies that support SLA description; examples are WSDL [26], WADL [27] and HATEOAS [28]. In the ServiceRegistry the SLA information shall be provided as meta-data. Technically it's stored in the DNS TXT field as key pairs. The following key pairs indicates how the SLA can be provided to a service consumer:

- *wadl=link*
- *wsdl=link*
- *hateoas=link*

The ServiceRegistry in addition can hold information regarding priorities within a System of Systems. The current approach to this is through the priority and weight fields of the DNS SRV record. This information is intended for use by Orchestration systems and quality of service systems to support automation QoS and dynamic re-organisation of automation operations based on QoS and service availability. The details of this, still a matter of further investigations.

For operator interaction with the ServiceRegistry, an optional MMI-ServiceRegistry system is defined, see Figure 10. This system provides a graphical user interface enabling the listing of published services.

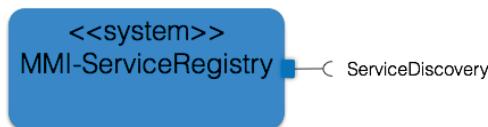


Figure 10: The optional MMI-ServiceRegistry system.

Authorization system (mandatory)

The objective of the Authorisation system is to provide Authentication, Authorisation and optionally Accounting of a system consuming a produced service. Based on the set of authorisation, authentication and optional accounting rules a service provider can ask whether a consumer is allowed to use a service resource or not.

Two different Authorisation systems are defined within the Arrowhead Framework. An AA - Authorisation Authentication system is defined. Next an AAA - Authorisation Authentication Accounting system is defined.

Based on existing technologies, the AA system is better suited for local clouds enrolling systems hosted on devices with sufficient computational power. While the AAA system is better suited for local clouds enrolling systems hosted on resource constrained devices. All Authorisation system graphs are color coded in red.

AA - Authorization system The AA Authorisation system implements an Authorisation system based on X.509 certificates [29]. It requires some computation power from a device and is thus not suitable for very resource constrained devices.

The Authorisation system provides two services and consumes one service, see Figure 11. The Authorisation system provides the ability to define and check the access rule for the consumption of services and its resources. Based on the access rules the service providers can ask whether a consumer is allowed to consume the service resource or not.

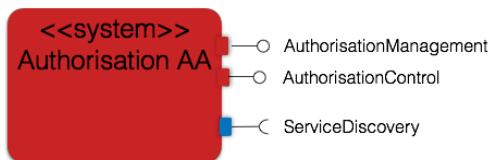


Figure 11: The mandatory Authorisation system, AA version.

The AuthorisationManagement service offers the possibility to manage the fine grained access rules for specific resources and also configure specific properties of the ticket like time-out. The AuthorisationControl service provides the possibility of controlling access to a service and a particular resource within the local cloud. This system includes both the authentication of the consuming system and the authorisation to consume a requested service. The system consumes the ServiceDiscovery service to publish the two produced services with the ServiceRegistry system.

AAA - Authorisation system The AAA — Authorisation system implements an Authorisation system based on Radius tickets [30]. This solution is feasible to apply in local cloud hosting resource constrained devices.

The ticket based AAA — Authorization system provides three services and consume the ServiceDiscovery service, see Figure 12.

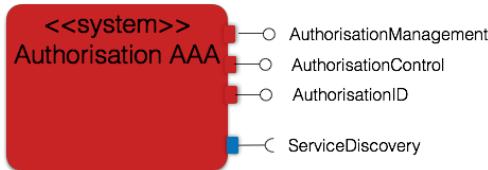


Figure 12: The mandatory ticket-based Authorisation system and its consumed and produced services.

The AuthorizationManagement service offers the possibility to manage the fine grained access rules for specific resources and also configure specific properties of the ticket like time-out. The AuthorizationControl service provides the possibility of controlling the access to a service and a particular resource within the local cloud. This system includes both the authentication of the consuming system and the authorisation to consume a requested service.

In addition to the services provided by the AA system, an AuthenticationID service is provided by the AAA system. There is a challenge-response mechanism to get a valid ticket to authenticate each consumer on the local cloud. The system consumes the ServiceDiscovery service to publish the three produced services with the ServiceRegistry system.

Orchestration system (mandatory)

The Orchestration system is a central component of the Arrowhead Framework and also in any SOA-based architecture [3]. In applications the use of SOA for a massive distributed System of Systems requires orchestration. It is utilised to dynamically allow the re-use of existing services and systems in order to create new services and functionality [31].

The process of orchestration is essential in support of service re-usability, service discoverability and service composability. From an architectural point of view, the Orchestration system is responsible for finding and pairing service consumers and providers. The input for this pairing is provided from, e.g., a plant description service, engineering tool, or an operator. In this regard, it provides advanced service discovery for systems. Orchestration is a key enabler for the engineering of Systems of Systems. Engineering information about suitable services, authorisation, and QoS is input to match-making algorithms and negotiation with the Authorisation and QoSManager systems.

The result of an orchestration request can vary over time, dependent on the environment and the requirements. This is because, in order to find the optimal service for the application system, there is much negotiation, with QoS for example, which may lead to different results. Therefore the orchestration must avoid oscillation in service selection.

Additionally within the scope of an local cloud it is strongly recommended that only a single Orchestration system is deployed.

The Orchestration system stores orchestration requirements and resulting orchestration rules. The requirements are specified using engineering tools, and provided to the Orchestration system either via a PlantDescription service or directly. New requirements can be provided in run-time upon which the Orchestration system computes new orchestration rules which are provided to the involved application systems. These rules consist of the endpoints where the produced service of interest is found.

In a local cloud, orchestration provides service consuming systems with service consumption patterns and endpoint information of the produced services to be consumed. Based on this information, the consuming system can request to consume the assigned service in an autonomous manner until a new orchestration is pushed to or pulled by a consuming system.

Some systems may have the capability to dynamically and/or temporarily create new service providers. For this purpose such system shall produce a service type called _ahf-servprod/_ahfc-servprod. An Orchestration system shall be able to consume such _ahf-servprod/_ahfc-servprod types and thus request the creation of such new service instances. In some cases there is a need to dynamically create new service providers in order to fulfil the System of Systems requirements, for example, in the case of the Arrowhead translator. Thus, while performing matchmaking, the Orchestrator system is able to consume the translation's _ahf-servprod/_ahfc-servprod service and instantiate a new translator for injection into the service consumption pattern.

Thus the objective of the Orchestration system is to provide a mechanism for distributing orchestration rules and service consumption patterns.

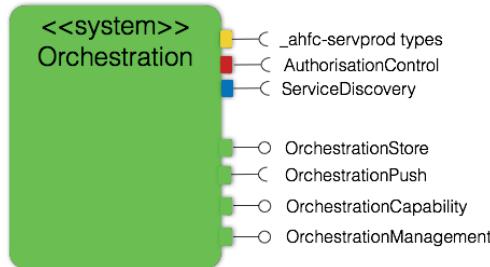


Figure 13: The orchestration system and its consumed and produced services.

The Orchestration system produces three services and consumes four services, cf. Figure 13. All Orchestration system graphs are color coded in green. The produced service, OrchestrationManagement, provides the possibility to manage the connection rules for specific services. Next the produced service, OrchestrationStore, provides the possibility for an application system to pull orchestration rules. Finally the produced

service OrchestrationCapabilities give the number of consumers currently consuming a specific service and which producers and consumers that are available for a certain service type. The service Orchestration Capability can be used for orchestration and/or to create a current state picture over the complete Arrowhead local cloud system-to-system interactions and information exchanges currently active.

The Orchestration system publishes its presence to the local cloud by consuming the ServiceDiscovery service from the ServiceRegistry system. It further shall be capable of consuming services of the type _ahf-servprod/_ahfc-servprod which then allows it to instantiate new service providers based on requirements and availability.

4.2 Automation support core systems

To facilitate automation application design, engineering, and operation, the Arrowhead Framework further contains a number of automation support services provided by the related automation support core systems. The objective of the automation support core systems is to support:

- The implementation of “plant” automation. Here plant could be the infrastructure of, e.g., a car manufacturing plant, a mine, an infrastructure
- Housekeeping within the local cloud
- Security and bootstrapping of a local cloud
- Inter-cloud service exchange
- System and service interoperability

For the purpose there are currently ten automation support core systems and their associated services defined. The systems currently defined are expected to address the architecture stated above. It's clear that additional automation support core systems and services will be defined in the future.

The automation support systems currently defined are

- PlantDescription system
- Configuration system
- DeviceRegistry system
- SystemRegistry system
- EventHandler system
- QoSManager system
- Historian system

- Gatekeeper system
- Translation system

These support systems are described below at a level corresponding to their respective SySD document. All support system graphs are color coded in yellow.

PlantDescription system

The objective of the PlantDescription system is to provide a basic common understanding of the layout of a “plant” or “site”, providing possibilities for actors with different interests and viewpoints to access their view of the same dataset provided by other sources.

Considering the different types of objects and relations that are expected to be present in a future Internet of Things (IoT) network, the concept of displaying different aspects of the same objects appears to be a useful solution to be able to present them all in an engineering tool.

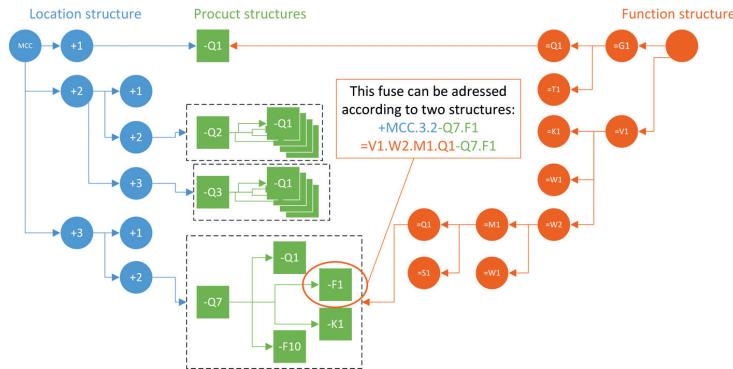


Figure 14: Example of how one component may be described by more than one viewpoint. Each colour in this diagram can be represented as a separate viewpoint by the plant description.

There are many different standards to describe automation objects. To keep a full database up to date would be an arduous if not impossible task, if the database is to contain all information that is required for all engineering tools and for all types of objects.

The approach of the PlantDescription system is instead to set up a separate system for identifying the objects and their relations while leaving the object details in the already established, standardised databases provided by existing engineering tools. Thus, throughout the design phase, all of the engineering data is still stored and maintained in the formats preferred by the engineering tools in their respective databases.

Figure 14 illustrates how a device or component documented according to IEC 81346 [32] may be found through traversing two different hierarchies, depending on the interest

of the user. Each colour in the figure can be represented as a separate viewpoint by the plant description, with blue representing the location of the object, red representing the function, and green representing the components making up a product. This standard and its approach to identifying objects was a major point of inspiration for the design of the PlantDescription system.

Further discussion of the design and prototype implementation of a PlantDescription system is provided in [33, 34].

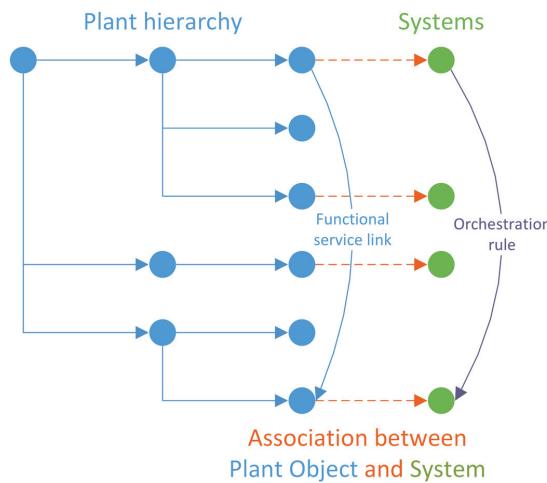


Figure 15: Illustration of how a relationship in the PlantDescription system can translate into an orchestration rule.

Figure 15 illustrates how a relationship between the logical objects in the PlantDescription system can be used to construct an orchestration rule, through the mapping of systems to plant objects. The “Functional service link” could, for example, be an internal link from an AutomationML structure, describing that the top object should send some data to the bottom object (where, e.g., the top object could be a sensor sending its data to a controller or actuator, represented by the bottom object). Such produced orchestration rules can then be provided to the Orchestration system via the OrchestrationManagement service.

The PlantDescription system produces three services and consumes the mandatory core services, as depicted in Figure 16.

The services produced are

- GetViewpoint

The service is used with an argument, Type, to get all nodes and links that correspond to this type.

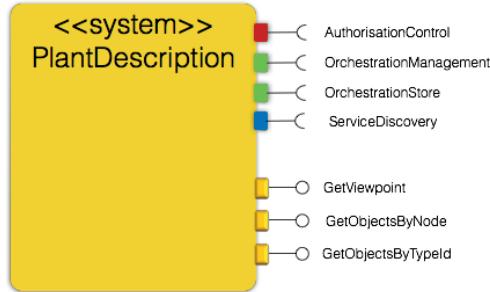


Figure 16: The PlantDescription system and its consumed and produced services.

- **GetObjectsByNode**

The service is used with a argument, NodeId, to get all nodes linked to the specified node, and those links.

- **GetObjectsByTypeid**

The service is used with a combination of argument of Type and NodeTypeid corresponding to the identity by which the node is identified according to that type. The function returns the same information as the GetObjectsByNode for that node.

The services consumed are

- OrchestrationStore
- OrchestrationManagement
- ServiceDiscovery
- AuthorisationControl

Configuration system

The Configuration system allows systematic management of configurable systems. Depending on the nature of the System of Systems the Configuration system may be limited to a store from which an application system may get its updated configuration or send a backup of its current configuration. In other scenarios the Configuration system may be able to compile configuration files from engineering data provided and manage which application systems should update their configuration at which time.

The objective of the configuration system is to enable systematic management of configuration information to configurable systems. The Configuration system shall be able to store and backup application configuration information. Such configuration information shall be possible to pull or push from the Configuration system. Typical configuration scenarios are

- Deployment of an Arrowhead Framework compliant software system to a device
- Control code to PLCs and IoT controllers
- Configuration of sensors and actuators - e.g., sample rate, sensitivity, filtering ...
- Configuration of HUIs

The Configuration system is producing one service and is consuming the mandatory core services, cf. Figure 17.

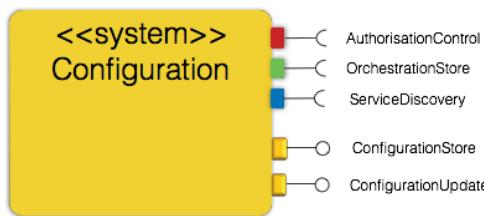


Figure 17: The Configuration system and its produced and consumed services.

The services produced are:

- ConfigurationStore
- ConfigurationUpdate

SystemRegistry system

The objective of the SystemRegistry system is to provide a local cloud storage holding the information on which systems are registered with a local cloud, meta-data of these registered systems and the services these systems are designed to consume.

The registration into a local cloud is part of the bootstrapping process of a local cloud, which is discussed in Section 4.4. The SystemRegistry system holds for the local cloud unique system identities for systems deployed within the Arrowhead Framework local cloud. This registry in combination with the DeviceRegistry is necessary to create a chain of trust from a hardware device to a hosted software system and its associated services.

The SystemRegistry shall in addition to registering the system identity also store

- Metadata about the system addressing non-functional information such as software revision, deployment info, etc. A full definition is found in Section 3.3.
- Services the system is designed to consume which includes data on

- ServiceTypes to consume: e.g., *_ahf-pidcontrol*
- SOA protocol capability: e.g., *http (REST)*
- Transport protocol: e.g., *tcp or udp*
- Payload data encoding: e.g., *JSON*
- Payload semantics: e.g., *sensML*
- Payload compression: e.g., *exi*
- Service interfaces, methods, and datatypes supported, e.g., hard coded based on IDD-... or through, e.g., *WADL-link* or *HETAOES-link*

In the current definition, the SystemRegistry system is producing one service and is consuming the mandatory core services, cf. Figure 18.

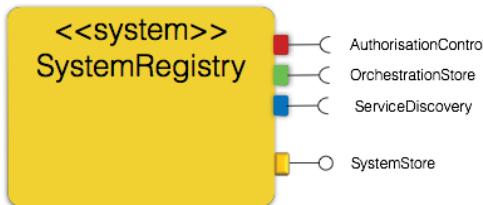


Figure 18: The SystemRegistry system and its produced and consumed services.

DeviceRegistry system

The objective of the DeviceRegistry system is to stores unique identities for devices deployed within an Arrowhead local cloud.

The DeviceRegistry system shall provide a local cloud storage holding information on which devices are registered with a local cloud. The registration into a local cloud is part of the bootstrapping process of a local cloud, which is discussed in Section 4.4. The DeviceRegistry system holds for the local cloud unique device identities for devices deployed. This registry in combination with the SystemRegistry is necessary to create a chain of trust from a hardware device to a hosted software system and its associated services.

The DeviceRegistry shall in addition to registering device identity also stor metadata about the device. The DeviceRegistry metadata is addressing non-functional information such as software revision, deployment info, etc.

The DeviceRegistry shall also hold data on which systems that are deployed to each registered device.

In the current definition, the DeviceRegistry system is producing one service and consumes the mandatory core services, cf. Figure 19.

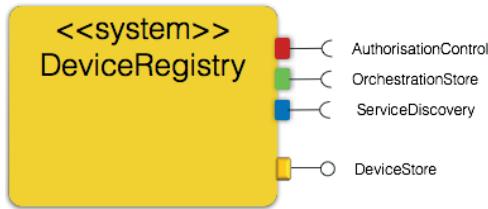


Figure 19: The DeviceRegistry system and its produced and consumed services.

EventHandler system

An event is an asynchronous occurrence. For a good event handling in a local cloud push and pull of services and the Publish - Subscribe approach should be supported. Not all systems may provide these capabilities. Further resource constrained devices may restrict the number of consumers to a certain service. For this purpose the Arrowhead Framework provides the EventHandler system. The EventHandler system [35] provides local cloud support for

- Event-based interaction for capability limited application systems
- Complex event filtering/processing
- Event logging
- Publish - subscribe functionality
- Service consumption buffer for capability/resource-limited application systems

For resource-limited systems not capable of publish and subscription, the EventHandler system provides a way of allowing for the publish and subscribe approach. The EventHandler will act as the service proxy and in addition provide the publish and subscribe capability. Since a subscription may be conditioned by some filtering rule, the EventHandler shall be capable of handling complex filtering rules. The EventHandler shall be able to handle multiple subscriptions to one service. Thus the EventHandler may receive events from service producers and dispatch them to one or several service consumers based on subscription conditions, cf. Figure 20.

The Orchestration system provides the orchestration rules informing the EventHandler which services to consume and which subscription filtering rules to apply. Thus enabled operations can be either explicit, by using the EventHandlerRegistry service, or implicit, by configuring the System of Systems in the three registries (DeviceRegistry, SystemRegistry, ServiceRegistry). In this latter declarative approach, event producers declare they consume the Publish service, event consumers declare that produce the Notify service, producers and consumers create rules on the plant description service regarding which events they produce or consume. The Orchestration system retrieves the information,

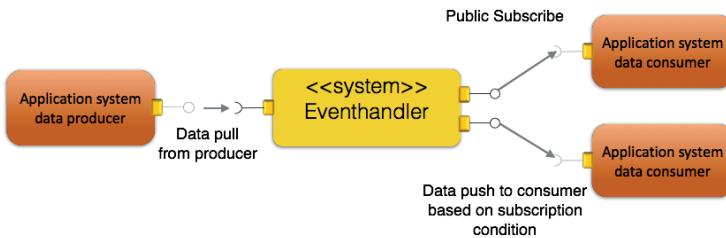


Figure 20: The EventHandler will consume events from a producing system and distribute these events based on the filtering requested by the subscribing system.

computes the matchings event producer, EventHandler instances, and event consumers and pushes the resulting orchestration rules to consumers and EventHandler instances.

The filtering rules applied to incoming events can be complex, e.g., based on message content, event metadata, and identity of publishers. Provided that the EventHandler has the computational and communication resources, it will off-load burden from systems residing on resource constrained devices, e.g., a cheap pressure sensor. Moreover, the EventHandler can be configured to log events to persistent storage by, consuming services from the Historian system; see Section 4.2.

Thus the objective of the Arrowhead Framework EventHandler system is to support the filtering and distribution of events, publish/subscribe communication, plus eventual storage of events and the associated data.

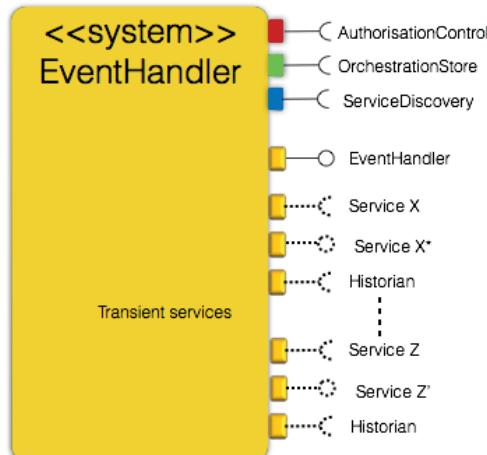


Figure 21: The EventHandler system and its produced and consumed services.

The EventHandler system produces the EventHandler service and consumes the mandatory core services, cf. Figure 21. The EventHandler service shall be consumed by the Orchestration system which initiates the creation of a transient pair of consumed and produced service X. To the consumed service X requested filtering and the subscription capability is applied. Which then is produced as service X_subscribe_filter. If requested the EventHandler also creates a transient service consumption of the Historian service, see Section 4.2.

QoSManager system

Quality of Service (QoS) within a local cloud is important. The automation requirements related to real-time communication and security have to be fulfilled. To achieve them, both monitoring of QoS and mitigation of QoS deviations shall be supported within a local cloud.

In the Arrowhead Framework architecture, the QoSManager system [36][37] will support QoS configuration and monitoring, in close collaboration with the Orchestration system.

Most of Arrowhead matchmaking between service producers and consumers is driven in a declarative manner: the Orchestration system interacts with DeviceRegistry, System-Registry, ServiceRegistry and PlantDescription systems to produce orchestration rules to individual application systems. Such orchestration data have to consider the QoS requirements set by individual application systems. These QoS requirements will be considered as constraints on the matchmaking.

The QoSSetup service will act as a support service to the Orchestration system. For every change in a local cloud, the resulting QoS has to be predicted. The changes cause the Orchestration system to compute alternative orchestrations which should be verified through the QoSSetup service. This will be repeated until a specific set of orchestrations appears to support the required QoS. Once the orchestration is settled the Orchestration system requests the QoSSetup service to perform the reservations necessary to grant the QoS. The Orchestration system also distributes the service end points to the systems involved.

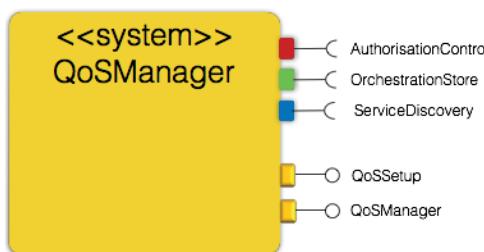


Figure 22: The QoSManager system and its produced and consumed services.

Thus, the QoSManager system's objective is to verify, manage, and guarantee QoS for services.

Apart from consuming the mandatory core services, the QoSManager system produces two services, QoSMonitor and QoSSetup, cf. Figure 22. In addition, the QoSManager consumes data from the SystemRegistry and the DeviceRegistry to deduce network topology and device capabilities within the local cloud. From this point of view, these registries provide information about the network tuning space. The current Arrowhead Framework view on QoS aspect and possible tuning spaces is further discussed in Section 3.5.

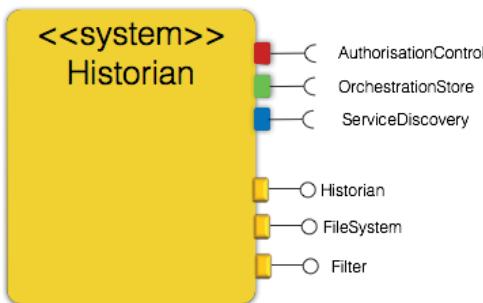


Figure 23: The Historian system and its produced and consumed services

Historian system

The objective of the historian is to provide on demand the possibility to log service exchanges and store and retrieve any payload data produced by services registered within the local cloud. Thus the Historian system provides the possibility to store audit information as well as to keep historical record of data produced within a local cloud. Service data and audit information can be extracted using filters. Thus, enabling the extraction of, for example, audit data regarding a producer and its activity period, number of payloads provided, errors, etc.

The Historian should be able to store any service events created by any application service in a local cloud. Which services to store events from is the responsibility of the Orchestration system. Two types of application systems can be distinguished

- Application systems capable of directly consuming the Historian service
- Application systems making used of the EventHandler system to interact with the Historian system

To extract data from the Historian is supported by two services:

- FileSys service

- Filter service

In summary, the Historian produces three services and consumes the mandatory core services, cf. Figure 23.

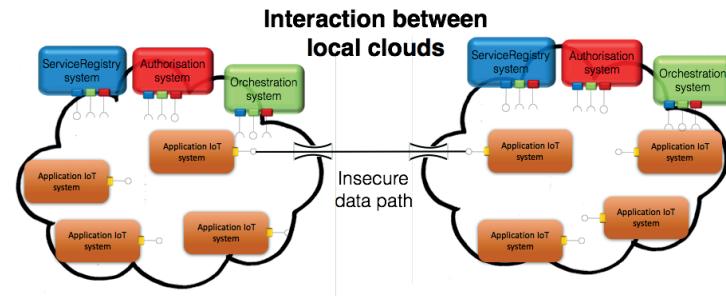


Figure 24: Service exchange between system residing in different local clouds may open an insecure data path. Such data a path punches a hole in the local cloud security wall, thus opening a path for intruders.

Gatekeeper system

Inter-cloud service exchange is essential to build real automation systems based on local clouds. Inter-cloud service exchange also supports scalability of a System of Systems.

For this purpose a local cloud need mechanisms that provide support for service discovery, system authentication and authorised service consumption, and data encryption. The scenario for establishing inter-cloud discovery, authentication & authorisation and orchestration visualised in Figure 25. Data encryption can still be maintained with e.g. IPSec or SOA protocol based encryption using for example MQTT; see the application discussion in Chapter 10.

The Arrowhead Framework approach to this is called the Gatekeeper system. The Gatekeeper system objective is to enable the discovery, orchestration, and authentication and authorisation of services residing in different local clouds.

The Gatekeeper system produces two services and consumes the mandatory core services of its local cloud, cf. Figure 26. A detailed description of the Gatekeeper system and its services can be found in Section 3.7.

For the inter-cloud service exchanges it's also necessary to secure the data path hole in the local cloud security fence. For security reasons it's not desirable that two systems residing in different local clouds should be able to directly exchange services.

The Arrowhead Framework provides two approaches to a secure data path. One approach makes use of the well-known Demilitarized Zone (DMZ) [38]. The approach is based on a double historian approach which is described in detail in Section 4.2. The

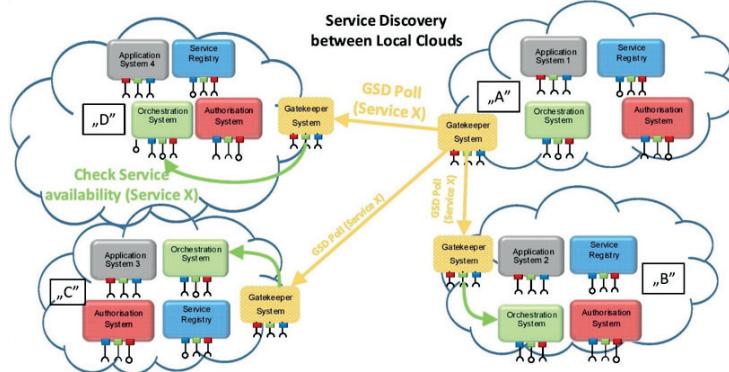


Figure 25: Inter-cloud service exchange supported by global service discovery, orchestration, and authorisation of service exchange.

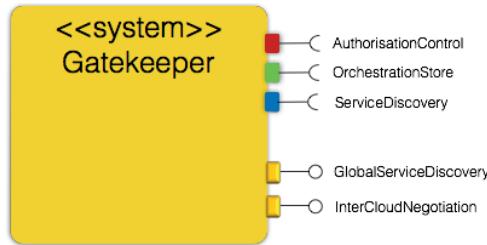


Figure 26: The Gatekeeper system and its produced and consumed services.

other approach, is based on the MQTT protocol and its broker technology; see a detailed description in Chapter 10.

Historian-Historian secure data path

When an inter-cloud service exchange has been orchestrated and authenticated using the Gatekeeper system the next step is establishing of a data path between the local clouds.

Establishing the inter-cloud data path can be done in several ways:

- Direct between two services residing in two different local clouds. This will provide a security hole in the local cloud and thus, an opening for exposing the local cloud to external communication and, e.g., a denial of service attack.
- To establish a secure data path Arrowhead Framework provides a type of demilitarized zone (DMZ) [38], solution. The buffering of application service data is made in the Historian system with an double DMZ layer approach as depicted in Figure 27. The DMZ devices hosting the two Historian system have to have dual

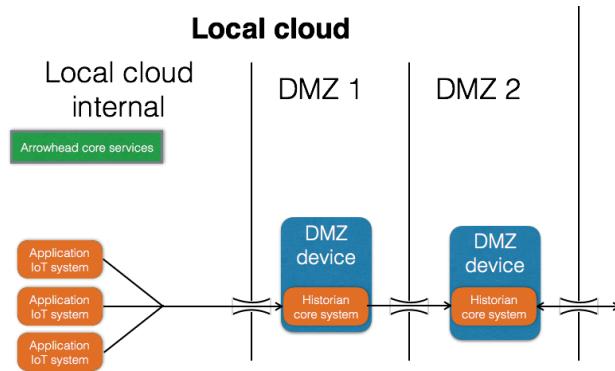


Figure 27: Establishing a secure inter-cloud data service patch by buffering data with a historian service which only can be accessed by another historian which in turn is exposed externally to the local cloud.

network interfaces thus, ensuring that no external traffic can reach the internal local cloud.

- Yet another way to establish the secure data path is by the use of the MQTT protocol and its broker technology; see Chapter 10 for details.

Translation system

The translation system is a transparency technology which resolves protocol, encoding, semantic, and security interoperability mismatches. An industrial IoT requires having multi-domain applications interacting seamlessly. Currently, industrial IoTs have many protocols which cannot communicate without specialised middleware or application layer multi-protocol interfaces. These solutions are both costly and not scalable. Hence an interoperability solution such as the translation system is required [39]. In Figure 28 a scenario with translation systems supporting interoperability between different services using incompatible SOA protocols is shown.

The objective of the Translation system is to provide translation of protocol, encoding, semantic and security between a service producer and a service consumer having non-interoperable SOA interfaces implementations.

The Translation system produces the Translation service and consumes the mandatory core services, cf. Figure 29. The Translation service shall be consumed by the Orchestration system which, initiates the creation of a transient pair of consumed and produced service X. The consumed service X uses protocol A and the produced service X uses protocol B.

Interoperability between protocols can be achieved in different ways. In Figure 30 there are three models of translator presented. In Figure 30-a the translation is made by direct translation. This model becomes very inefficient as the number of protocols

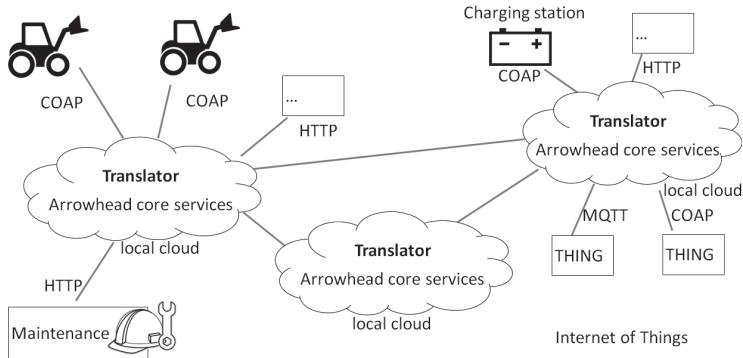


Figure 28: Service exchange scenario with translation systems supporting the interoperability between different services using different SOA protocols. Here protocol translation is supporting within a local cloud and in-between local clouds

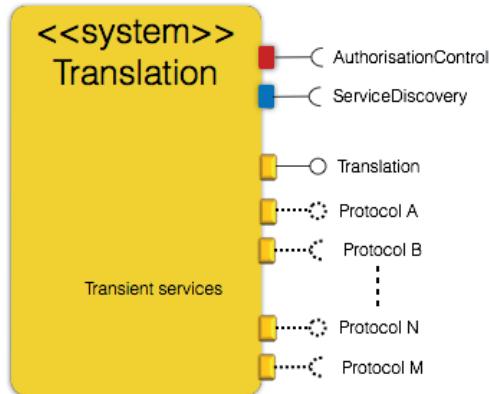


Figure 29: The Translation system and its produced and consumed services.

increases. The number of required translators is given by Equation 1.

$$\sum_{k=1}^{n-1} k = \frac{n(n - 1)}{2} \quad (1)$$

In Figure 30-b the translation is using an intermediate protocol, this is much more efficient in terms of translation implementations. However, it introduces latency and potentially additional loss of information. In 30-c translation uses an intermediate format, not constrained by the requirements of on-the-wire protocols, the translator is able to scale well and also maintain very good latency performance and information preservation.

Thus the Arrowhead Framework currently has defined a translation system according to each of the above given core systems; their services and interactions are described in

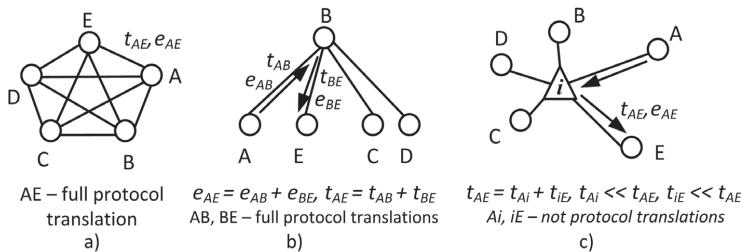


Figure 30: Three different translation models possible in a protocol translator system.

detail in Chapter I.

4.3 Application systems

Within the Arrowhead Framework the application system objectives are to implement application functionalities and services aiming to fulfil application requirements.

An application system may produce or consume other Arrowhead Framework services.

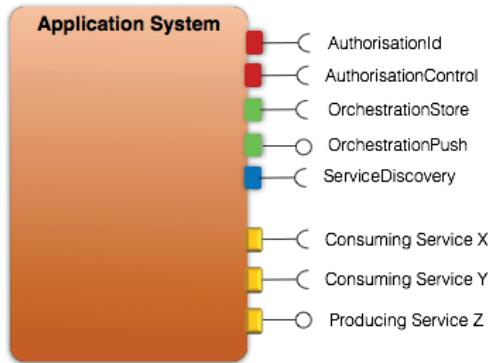


Figure 31: An application system is capable of consuming the Arrowhead Framework mandatory core services and will produce and/or consume one or more application services

Application services

An application system is at minimum consuming the following mandatory core services; cf. Figure 31:

- ServiceDiscovery
- AuthorisationControl

- OrchestrationStore

In addition it is producing or consuming at least one service. The application service may be capable of handling a service publish and subscribe schema. This shall be provided as metadata when registering the service with the ServiceRegistry.

Regarding use of authorisation the application shall provide meta-data if it can interact with an AA or an AAA authorisation system.

The application system may be capable of using the Historian system for data logging and audit information. This can be accomplished by an orchestration rule to the Historian system to consume the produced application service. For systems that are resource constrained and have sleeping functionalities, it might be feasible for the application system to have the capability to consume the Historian service from the Historian system. Thus minimising administration of awake time slots and the uncertainties that missed time slots may create.

4.4 Deployment procedure for a local cloud

The Arrowhead Framework has architectural components that address the initiation of a local cloud and the invocation of trusted devices.

The objective of the deployment procedure is to create a local cloud in such a way that its basic functionalities and security have been established in a secure way. It is obvious that the deployment shall ensure that the mandatory core systems, their services and executing devices can be guaranteed to not be compromised.

Regarding enrolment of devices, systems and services into a local cloud there are bootstrapping mechanisms defined below. Of course a non-secure approach can be used here and just allow any device and its systems to register service into a local cloud. In most cases it is good practise to have a process that authenticates and authorises the entry of devices, systems, and services into a local cloud.

Secure bootstrapping of devices and systems into a local cloud

Assure that the ServiceRegistry system, the DeviceRegistry system, the SystemRegistry system, the Authorisation system, and the Orchestration system are established in a network in a secure way. Have these, non-compromised, systems executing on some devices. Thus the mandatory systems of a local cloud are established. To build a secure automation application the bootstrapping of application devices and systems, plus eventual support core systems and related devices, is critical from a security point of view.

To assure that the cloud is not compromised upon the introduction of systems, it is important to establish a chain of trust from service to system to device. For this purpose a secured initiation or bootstrapping process ranging from a device to its software system and associated services is needed. For the Arrowhead Framework, the proposed approach is based on a two-way clearance procedure of a device and its hosted software systems. The approach is based on a two way authentication.

To enable a device to be trusted, it has to have specific hardware providing storage and computation of authentication keys which shall be secure and tamper free. Such hardware, security controllers, are provided by a couple of vendors. The device further needs both a network interface and some type of short range communication, e.g., NFC. This will enable an operator identification of a device via key authentication over the NFC link. Such authentication will allow the generation of a device authentication key to be transferred to and stored in the security controller. The same procedure is to be made for each Arrowhead Framework system and each service produced by the system service. Thus the security controller holds the chain of trust including device key, system key and service key.

An Arrowhead Framework compliant system can then request to join a local cloud by requesting to

- register with the DeviceRegistry providing its device credential
- register with the SystemRegistry providing its system credential

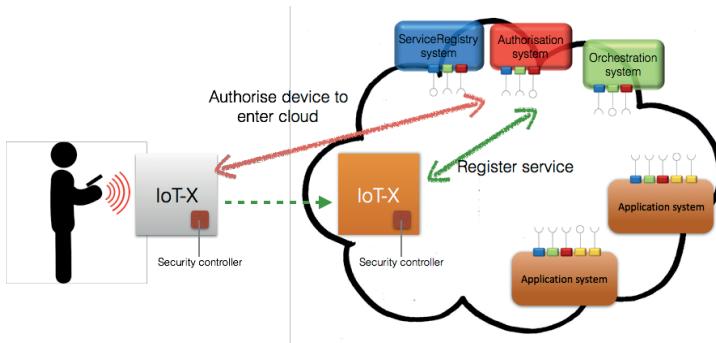


Figure 32: The bootstrapping process for a device, IoT-X, with IoT system A requesting to be entered into an Arrowhead Framework local cloud. A two-way authentication process is defined, involving human authorisation of the device and its hosted systems. Upon successful identification, the security controller in the device is provided with the keys allowing the Authorisation system to identify and admit the device and its systems into the local cloud.

These credentials, key plus ID, will now be used for identification with the Authorisation system within the local cloud. First the device key is authenticated upon which the local cloud membership is authorised by the Authorisation system. The Authorisation system in response provides the local cloud membership key to the device. Next each system hosted by the device is requesting membership of the local cloud using the same schema. This process is then repeated for each device and system to be registered with the local cloud. Finally all services produced by the registered systems shall be authenticated and registered with the ServiceRegistry system. This process of becoming a trusted

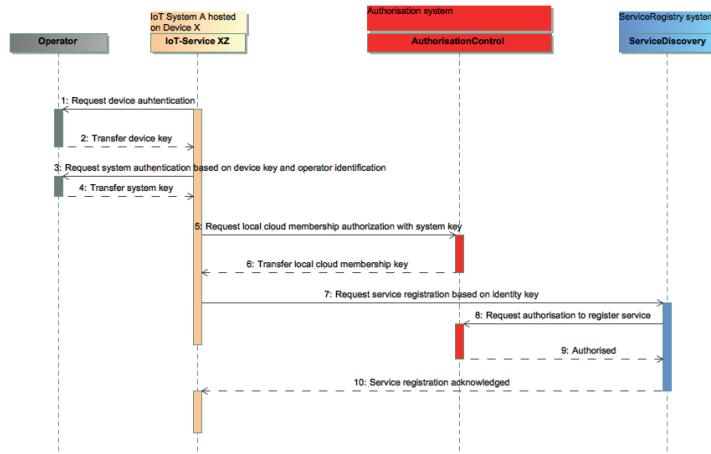


Figure 33: The sequence diagram for the process of authenticating a service (or system or device) with its registry thus allowing the registration of the service (or system or device) with its registry.

device and system/s with trusted service/s within a local cloud is depicted in Figure 32. A sequence diagram for authenticated registration of a service with the ServiceRegistry is given in Figure 33. The sequence diagrams for a device and a system registering with their Registry are identical. In this way any device, system and associated service will be authenticated with a local cloud and a chain of trust is established.

The operator authentication approach should also be applied to the bootstrapping of the three mandatory systems plus the SystemRegistry and DeviceRegistry systems. Thus, providing a first level trust to these very core parts of a local cloud. Here the security controller within the device/s hosting these five systems and their services will store the operator interaction generated key together with the respective IDs.

4.5 Creating Arrowhead Framework compliant systems

When designing an application system to operate within the Arrowhead Framework architecture devices, systems and services have to comply to the architecture principles. Following certain design steps will help to design an Arrowhead Framework compliant system and associated services. The following steps shall be performed to ensure compliance to the Arrowhead Framework:

- Design according to Arrowhead Framework templates;
- Adapt legacy systems or implement new systems according to the Arrowhead Framework principles/patterns;
- Perform interoperability tests.

Besides the available design templates, the work of system architects and developers is supported through the design documentation guidelines defined in [19].

System design is often created by using already available legacy system components. In order to support interoperability for legacy systems, these should be adapted to the Arrowhead Framework. This can be done through adaptor or gateway components, which can be either integrated to the legacy systems (i.e., within the same hardware element), or can be made available as an Arrowhead core system with such gateway/adaptor capabilities.

To support interoperability testing, Arrowhead Framework provides a test tool which is further described in Chapter 5.

4.6 Interfacing to legacy systems

Due to the large amount of automation technology already installed there is a clear need for migration strategy and technology allowing the integration of the local cloud approach to legacy systems. The Arrowhead Framework automation architecture defines three levels of maturity. The maturity level indicates in what way an application system achieves conformance with the Arrowhead Framework; cf Figure 34. Three levels are defined:

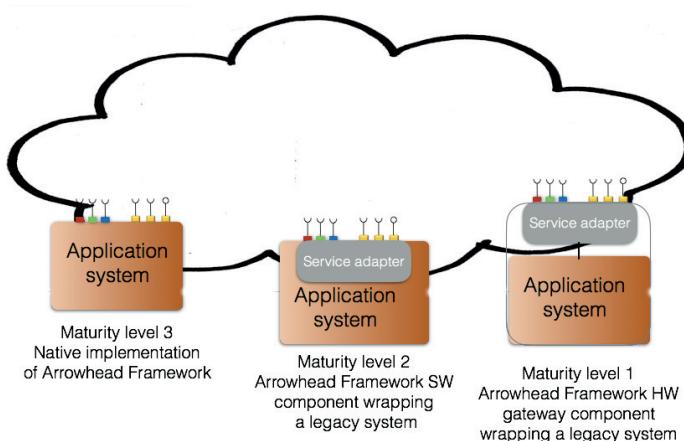


Figure 34: Application systems must publish information about their available services.

- **Level-3** The application system implements the consumption (AND/OR) production of Arrowhead Framework compliant services without using any external components.
- **Level-2** The application system implements the consumption (AND/OR) produc-

tion of Arrowhead Framework services by using a software adaptor. The application system is thus modified by integrating the software adaptor into the system.

- **Level-1** The application remains unchanged. It uses dedicated hardware with software responsible for wrapping the application system with Arrowhead Framework compliant services. This hardware/software system implements a specific interface that can be connected to an existing system interface and proxies this information and functionality to Arrowhead Framework compliant services. The existing application system can remain unchanged.

4.7 Verification of Arrowhead Compliance

The purpose of the verification is to make sure that the system in question is compliant to the Arrowhead framework. The following is checked during the verification procedure.

- Can it connect and communicate properly with the mandatory core services?
- Does it comply with the rules for system documentation set for Arrowhead Framework compliant systems?
- Does it produce and consume services of the Arrowhead Framework as it is documented within its System Description (SysD)?

In order to technically validate the compliance, the Arrowhead Verification Tool has been created. It supports the following:

- System test and integration procedures through manual, automatic, and scripttests in order to verify and validate service realisation;
- Development through manual orchestration — to simplify producer and/or consumer interaction (with functionalities such as recording and playback of service interactions);
- Dynamic simulation of services and the handling of function chains, as well as service relations and their information exchange.

The compliance verification is to be controlled by personnel to evaluate operational as well as documentation aspects within different scenarios.

In conclusion, the Arrowhead Framework consists of what is needed for anyone to design, implement and deploy an Arrowhead compliant system aiming at enabling all of its users to work in a common and unified approach – leading towards high levels of interoperability, supporting the general objective of enabling information exchange between two IoT devices at a service level. Also further supporting the objective of enabling System of Systems operation. In turn enabling design, engineering and operation of collaborative automation systems using a local cloud approach addressing key properties of real time, security, engineering, and scalability.

For latest update on the Arrowhead Framework and the local automation cloud architecture, please consult the Arrowhead Framework wiki [25].

References

- [1] 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*, 2014.
- [2] S. Karnouskos, A. W. Colombo, F. Jammes, J. Delsing, and T. Bangemann, "Towards an architecture for service-oriented process monitoring and control," in *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2010, pp. 1385–1391.
- [3] T. Erl, *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007.
- [4] C. Pautasso, E. Wilde, and R. Alarcon, Eds., *REST: Advanced Research Topics and Practical Applications*. Springer, 2014.
- [5] Z. Shelby, "The constrained application protocol (coap) - rfc 7252," IETF, Tech. Rep., 2014.
- [6] "Xmpp is the open standard for messaging and presence," 2016. [Online]. Available: <http://xmpp.org>
- [7] (2016) Mqtt is a machine-to-machine (m2m)/"internet of things" connectivity protocol. [Online]. Available: <http://mqtt.org>
- [8] A. Banks and R. Gupta. (2016) Mqtt version 3.1.1. oasis standard. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>.
- [9] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer, 2009.
- [10] S. Cheshire and M. Krochmal, "Dns-based service discovery," IETF, Tech. Rep., 2013.
- [11] P. Mockapetris, "Domain names - implementation and specification," IETF, Tech. Rep., 1987.
- [12] R. Elz and R. Bush, "Clarifications to the dns specification," IETF, Tech. Rep., 1997.
- [13] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry," IETF, Tech. Rep., 2011.
- [14] Wikipedia, "User datagram protocol — wikipedia, the free encyclopedia," 2016, accessed 20-April-2016. [Online]. Available: https://en.wikipedia.org/w/index.php?title=User_Datagram_Protocol&oldid=715583050

- [15] J. Postel, “Rfc 768 user datagram protocol,” IETF, Tech. Rep., 1980.
- [16] “Extensible markup language - xml,” 2016. [Online]. Available: <https://en.wikipedia.org/wiki/XML>
- [17] D. Peintner and S. Pericas-Geertsen, “Efficient xml interchange (exi) primer,” W3C, Tech. Rep., 2014.
- [18] C. Jennings and Z. Shelby, “Media types for sensor markup language (senml) draft-jennings-senml-10,” IETF, Tech. Rep., 2013.
- [19] F. Blomstedt, L. L. Ferreira, M. Klisics, C. Chrysoulas, I. M. de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, “The arrowhead approach for soa application development and documentation,” in *Proceedings IECON 2014*, 2014.
- [20] “Arrowhead project.” [Online]. Available: <http://www.arrowhead.eu>
- [21] Wikipedia, “Unified modeling language — wikipedia, the free encyclopedia,” 2016, [Online; accessed 19-April-2016]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Unified_Modeling_Language&oldid=708110201
- [22] ——, “Business process model and notation — wikipedia, the free encyclopedia,” 2016, [Online; accessed 19-April-2016]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Business_Process_Model_and_Notation&oldid=715814821
- [23] ——, “Systems modeling language — wikipedia, the free encyclopedia,” 2016, [Online; accessed 19-April-2016]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Systems_Modeling_Language&oldid=715050337
- [24] ——, “Automationml — wikipedia, the free encyclopedia,” 2015, [Online; accessed 19-April-2016]. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=AutomationML&oldid=673745347>
- [25] (2016) Arrowhead framework wiki. [Online]. Available: https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page
- [26] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, “Web services description language (wsdl) 1.1,” W3C, Tech. Rep., 2001.
- [27] M. Hadley, “Web application description language,” W3C, Tech. Rep., 2009.
- [28] Wikipedia, “Hateoas — wikipedia, the free encyclopedia,” 2016, [Online; accessed 2-June-2016]. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=HATEOAS&oldid=715478952>
- [29] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, “X. 509 internet public key infrastructure online certificate status protocol-ocsp,” RFC 2560, Tech. Rep., 1999.

-
- [30] A. DeKok and A. Lior, “Remote authentication dial in user service (radius) protocol extensions,” RFC 6929, April, Tech. Rep., 2013.
 - [31] K. Nagorny, R. Harrison, A. W. Colombo, and G. Kreutz, “A formal engineering approach for control and monitoring systems in a service-oriented environment,” in *11th IEEE International Conference on Industrial Informatics (INDIN)*, July 2013, pp. 480–487.
 - [32] “Iec81364 - industrial systems, installations and equipment and industrial products – structuring principles and reference designations,” ISO/IEC, Standard, 2009.
 - [33] O. Carlsson, D. Vera, J. Delsing, B. Ahmad, and R. Harrison, “Plant descriptions for engineering tool interoperability,” in *Proceedings of IEEE INDIN 2016*, 2016.
 - [34] O. Carlsson, C. Hegedűs, J. Delsing, and P. Varga, “Organizing iot systems-of-systems from standardized engineering data,” in *Proceeding IECON 2016*, Firenze, Italy, Oct. 2016.
 - [35] M. Albano, L. Ferreira, and J. Sousa, “Extending publish/subscribe mechanisms to soa applications.” in *Proceedings of the 12th IEEE World Conference on Factory Communication Systems (WFCS)*, Aveiro, Portugal, May 2016.
 - [36] M. Albano, R. Garibay-Martínez, and L. L. Ferreira, “Architecture to support quality of service in arrowhead systems,” in *Proceedings of INFORUM 2015*, Covilhã, Portugal, Sep. 2015.
 - [37] L. L. Ferreira, M. Albano, and J. Delsing, “Qos-as-a-service in the local cloud,” in *Proceedings of SOCNE 2016, in conjunction with ETFA 2016*, Berlin, Germany, Sep. 2016.
 - [38] S. Jacobs, *Engineering Information Security: The Application of Systems Engineering Concepts to Achieve Information Assurance*. John Wiley & Sons, 2015, no. ISBN 9781119101604.
 - [39] H. Derhamy, J. Eliasson, J. Delsing, P. Varga, and P. Punal, “Translation error handling for multi-protocol soa systems,” in *Proceedings of 2015 IEEE 20th International Conference on Emerging Technologies & Factory Automation (ETFA 2015)*, Luxembourg, Sept. 2015.

PAPER I

Arrowhead Framework core systems and services

Authors:

Jerker Delsing, Jens Eliasson, Michele Albano, Pal Varga, Luis Ferreira, Hasan Derhamy, Csaba Hegedűs, Pablo Puñal Pereira, and Oscar Carlsson

Reformatted version of paper originally published in:

Book chapter, IoT Automation: Arrowhead Framework, Taylor & Francis, 2016.

© Taylor & Francis 2017 From IoT Automation: Arrowhead Framework by Oscar Carlsson, author Jerker Delsing, editor. Reproduced by permission of Taylor and Francis Group, LLC, a division of Informa plc.

Chapter 4 - Arrowhead Framework core systems and services, 2017, Jerker Delsing, Jens Eliasson, Michele Albano, Pal Varga, Luis Ferreira, Hasan Derhamy, Csaba Hegedűs and Pablo Puñal Pereira, *Arrowhead Framework core systems and services*, Taylor & Francis, 2017.

1 Introduction

In Chapter 2 local clouds were discussed followed by a local cloud automation architecture in Chapter 3. The automation architecture supports the implementation of local automation clouds. Such implementation is supported by the Arrowhead Framework and its core systems and services.

The Arrowhead Framework core systems enable the creation and operation of local clouds. First implementation of these systems and their services are described in detail in this chapter.

There are currently two types of core services within the Arrowhead Framework:

- Mandatory core systems
needed to establish the minimal local cloud.
- Automation support core systems
extending local cloud capabilities intending to provide support for the design and operation of local automation clouds and interaction between local clouds.

2 Mandatory core systems and services

Mandatory core systems provide the minimum advisable services to establish a local automation cloud. The mandatory core systems are

- ServiceDiscovery system — Responsible for registering and enabling discovery of registered services.
- Authorisation system — Responsible for providing credentials to systems in the local cloud enabling system authentication and service exchange authorisation.
- Orchestration system — Responsible for providing service consumption patterns information to the system registered in the local cloud.

These mandatory core services will support the fundamental architectural properties assigned to a local cloud, as discussed in Chapter 3.

The Arrowhead Framework implementation of the mandatory core systems and their services provides these fundamental properties to a local cloud. Thus, enabling and allowing service exchanges between a service producer and a service consumer with the desired level of security and autonomy, which is briefly illustrated in Figure 1.

2.1 ServiceRegistry system

The Arrowhead Framework provides an implementation of the ServiceRegistry system. Implementing one of the three mandatory architectural systems to enable a local automation cloud. The ServiceRegistry system and its ServiceDiscovery service enable publication, lookup, and deletion of services in the service registry. The underlaying

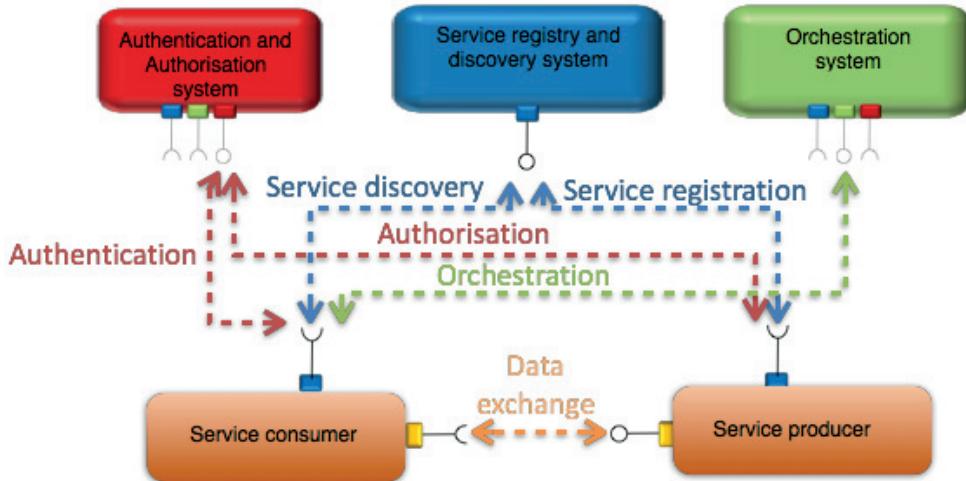


Figure 1: Minimal local cloud with indications of mandatory core service interaction enabling service exchange between two application systems.

technology is DNS with the DNS-SD extension [1][2][3][4][5]. This adheres to well-known and proven Internet standards and technology.

The ServiceRegistry system is capable of storing reference information to all active producing services within the local cloud where it is active.

All systems within the local cloud that have services that produce information shall publish their produced service with the ServiceRegistry system by using the ServiceDiscovery service. A registration time out is recommended. For systems having sleep periods such meta data have to be provided. The DNS TXT record provide the location for time out and meta-data. The ServiceRegistry system uses this information to check for service availability in the local cloud. No response may result in service de-registration.

ServiceDiscovery service

The ServiceDiscovery service and its interface is defined according to Figure 2. The service interface provides three methods

- Publish

The publish method is used to register services. The services will contain a symbolic name as well as a physical endpoint. The instance parameter represents the endpoint information that should be registered.

- Un-publish

The un-publish method is used to unregister a service that no longer should be used. The instance parameter contains information necessary to find the service to be removed.

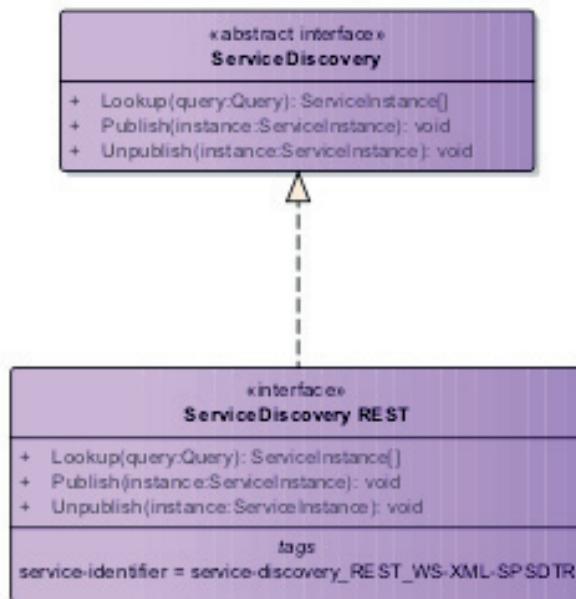


Figure 2: *ServiceRegistry* system produces the *ServiceDiscovery* service with the here depicted interfaces.

- **Lookup**

The lookup method is used to find and translate a symbolic service name into a physical endpoint, IP address, and a port. The query parameter is used to request a subset of all the registered services fulfilling the demand of the requesting system. The returned listing contains service endpoints that fulfils the query.

The lookup, publish, and un-publish method sequences are provided in Figures 3, 4, and 5.

The information model follows the definitions in Section 2.6 and holds two data types:

- **ServiceRecord**

with the following data:

- Endpoint - string

This datatype implements a representation of an endpoint using DNS A-records:

- * Hostname is a String containing the name of the host in format:
name.domain.topdomain, e.g., *app.arrowhead.eu*
- * path: 192.168.1.20
- * Port is an Integer containing the port number, e.g., 8070.

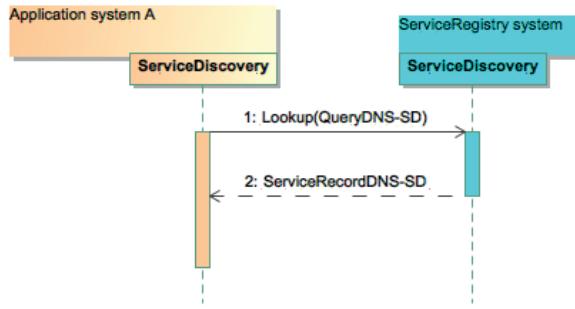


Figure 3: Sequence diagram for the lookup method of the ServiceDiscovery service.

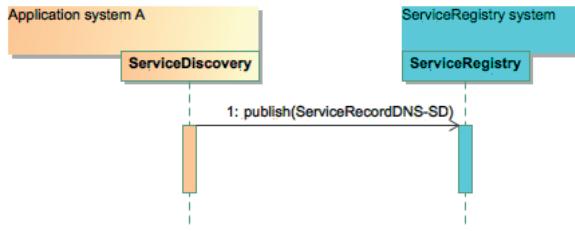


Figure 4: Sequence diagram for the publish method of the ServiceDiscovery service.

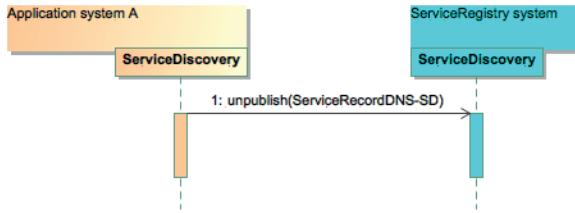


Figure 5: Sequence diagram for the un-publish method of the ServiceDiscovery service.

- * Edata is a String containing additional information related to the endpoint. Any additional information that is required to identify the service instance should be stored in the mandatory DNS TXT record, as discussed in Section 2.6 and defined in [3].
- * Metadata: key=value
Metadata for the service are stored as key value pairs, could be, e.g., time to live, sleep period, configurations, payload encoding, compression and semantics. To allow for the orchestration to understand if any translation

is necessary, it is proposed that the following three be mandatory:

- Encoding: e.g., *encode=xml* where XML [6] encoding is used and specified in the CP (Communication Profile) document
 - Compression: e.g., *comp=exi* when EXI [7] compression is used and specified in the CP document
 - Semantics: e.g., *sem=senml* where SenML [8] semantics is used and specified in the SP (Semantic Profile) document
- * ServiceName - string
Name of the service instance e.g. *_Temp1*
- * ServiceType - string
e.g. *_ahf-temperature._coap._udp.*

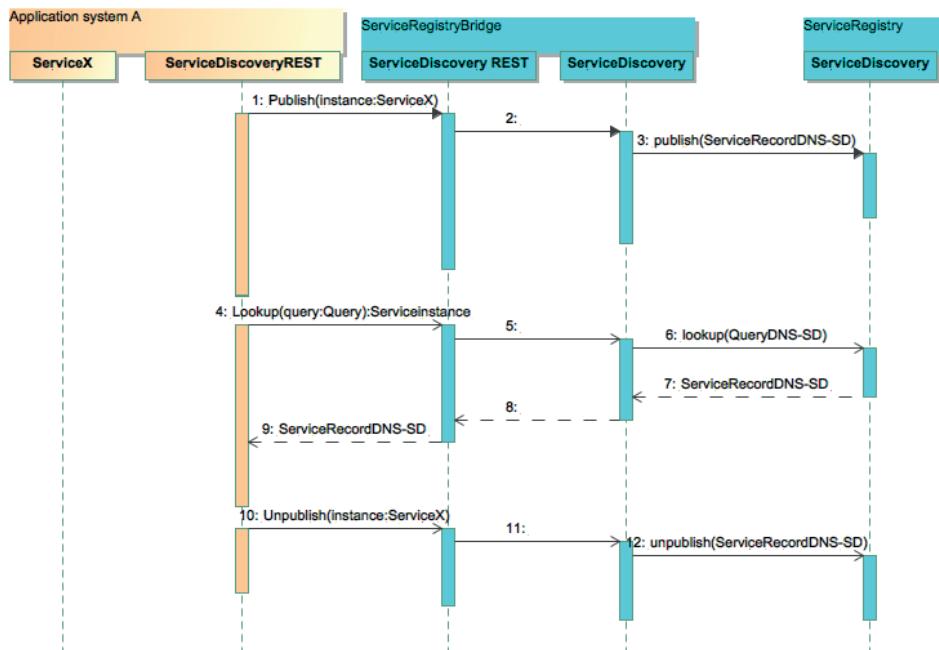


Figure 6: Sequence diagram a REST based system A registration process with the ServiceRegistry as supported by the ServiceRegistryBridge.

- Query
with the following content:
 - Query — string

where the query string specifies one or several of the data types of the service. The query will then return a list of all registered services with the specified data type/s.

To access the ServiceRegistry from a REST based system, Arrowhead Framework has implemented the ServiceRegistryBridge system allowing easy ServiceRegistry interaction from a REST — http protocol based system.

The sequence diagram for REST based system interactions with the ServiceRegistry is shown in Figure 6.

The details of SLA meta-data and priority information in the service registry are beyond the scope of this book. Please refer to the Arrowhead Framework wiki for the latest details.

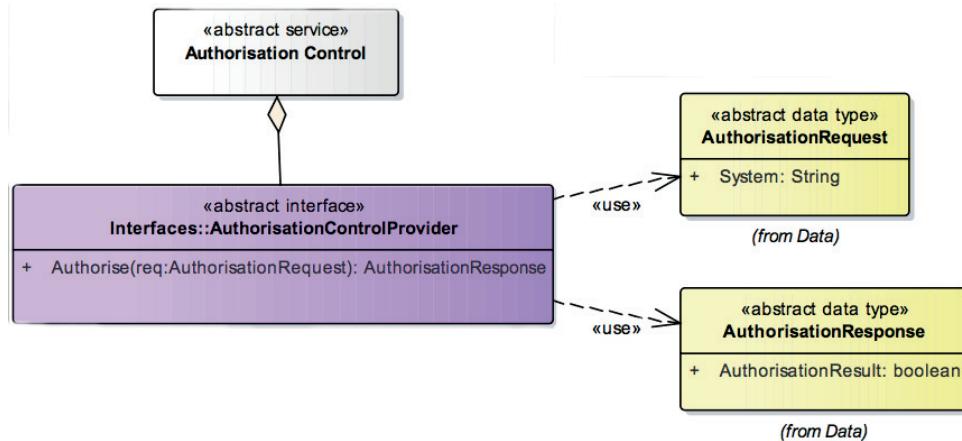


Figure 7: The AuthorisationControl interface with available methods and datatypes.

2.2 Authorization system

The Arrowhead Framework implements two different Authorisation systems. One is a authorisation and authentication, AA, system. The other is a authorisation, authentication and accounting, AAA, system. Both the AA and the AAA system meet the basic objectives requested by the Arrowhead Framework local cloud architecture.

AA-Authorisation system

Here an authorisation and authentication, AA, system is discussed. The AA-Authorisation system produces the AuthorisationControl service and the AuthorisationManagement

service.

For the AuthorisationControl service the available interface is AuthoriseControlProvider; see Figure 7.

The Authorise Control Provider interface provides a clearance for a specific system to consume a specific service. The method is

- Authorise
 - with the data types:
 - AuthorisationRequest
 - requests the authorisation for the system addressed by an endpoint string e.g. `coap://192.168.2.30:8000/_viib3._ahf-vibration._udp`. The end point as provided by the orchestration system.
 - AuthorisationResponse
 - responds with a boolean (True/False) request.

A sequence diagram for an application system A requesting authorisation to consume a specific method of a produced service is provided in Figure 8.

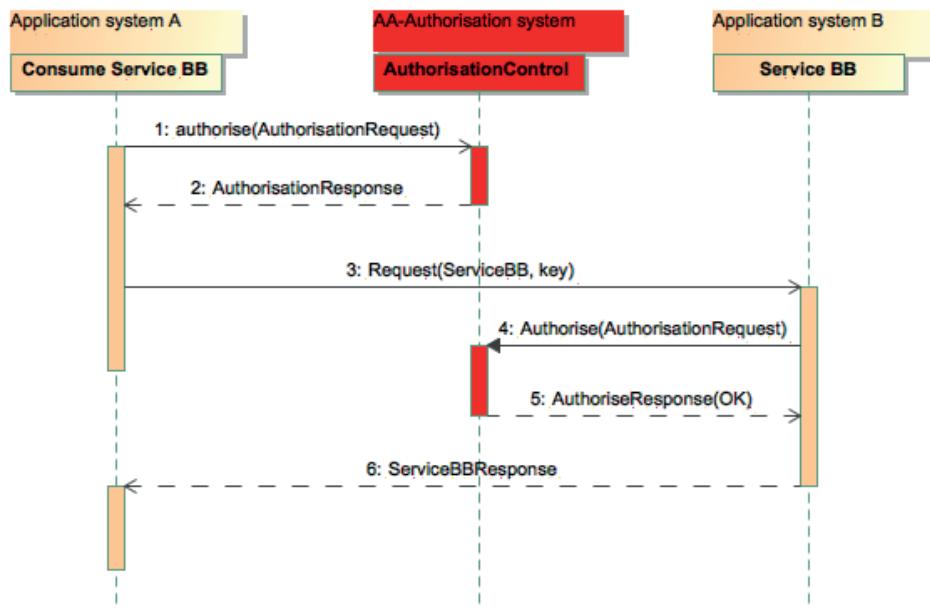


Figure 8: The sequence diagram for system A requesting authorisation to consume the interface definition and available data types of the AuthorisationManagement service.

For the AuthorisationManagement service the available interface is AuthoriseManagementProvider; see Figure 9.

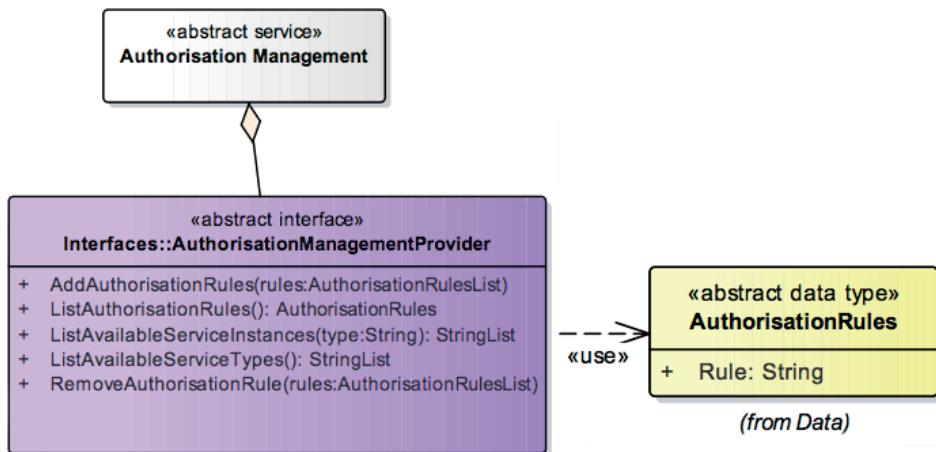


Figure 9: The AuthorisationManagement interface with available methods and data types.

The AuthoriseManagementProvider interface provides a number of methods to manage authorisation rules:

- `AddAuthorisationRules(rules:AuthorisationRulesList)`
is used to store new authorisation rules in the providing system.
- `ListAuthorisationRules(): AuthorisationRules`
is used to present all the rules to the administrator of the authorisation ruling.
- `ListAvailableServiceInstances(type:String): StringList`
is used to fetch all service instances that currently are stored in the authorisation system, in order to list consumer systems and producer systems. The purpose of this method is to allow an administrator to administer all systems.
- `ListAvailableServiceTypes(): StringList`
is used to fetch the service types currently used in the arrowhead system-of-system, from the authorisation point of view.
- `RemoveAuthorisationRule(rules:AuthorisationRulesList)` is used to remove a rule that no longer is valid.

with the datatype:

- AuthorisationRules

The AuthorisationRules data type contains information about rules that the consumer should be matched against in order to determine if a producer should be releasing information.

Sequence diagrams for the methods listAuthorisationRules and addAuthorisationRule are provided in Figure 10 and 11. Sequence diagrams for the other methods can easily be deduced based on these sequence diagrams.

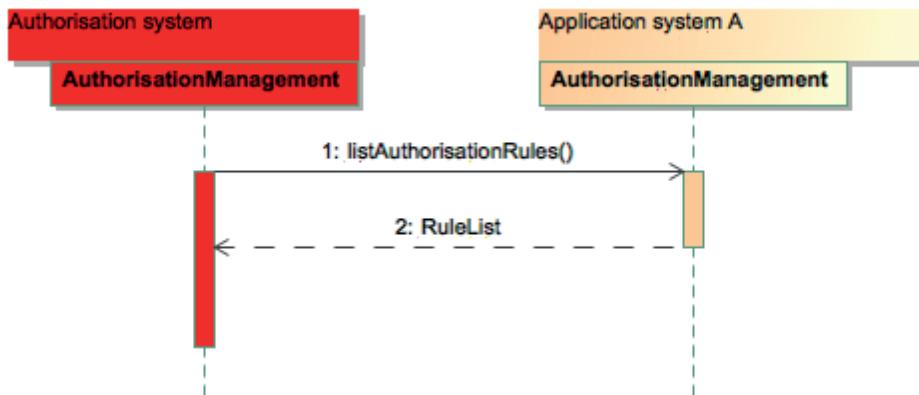


Figure 10: Sequence diagram for the list AuthorisationRule method of the AuthorisationManagement service providing a list of AuthorisationRules applicable to a system A.

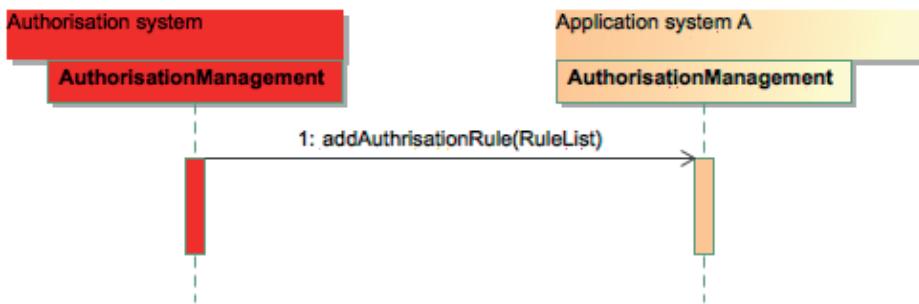


Figure 11: Sequence diagram for the addAuthorisationRule method of the AuthorisationManagement service providing AuthorisationRules to a system A.

The most recent information on the AA-authorisation system is available in the Arrowhead Framework wiki [9].

AAA-Authorization system

Here an authorisation, authentication, and accounting, AAA system is discussed. The Arrowhead Framework implementation of the AAA-Authorisation system and is based on Radius ticket technology.

The published services are

- AuthenticationID service
 - providing the possibility to complete a Challenge-Response communication with the Authentication, Authorization and Accounting Server to get a new and an unique Ticket.
- AuthorisationControl service
 - providing the possibility of enabling fine grained access control to any resource/service for external requests; also provides customised information about the external consumer.
- AuthorisationManagement service
 - providing the possibility to manage the access control policies, accounts, accounting parameters, timeouts, etc.

The available interface of AuthenticationID is AuthenticationIDProvider, see Figure 12.

The AuthenticationIDProvider interface provides a resource for a new Ticket request based on a Challenge-Response. The method is

- Authenticate ID with the data types
 - AuthenticationRequest
 - request a new and valid authenticator
 - ChallengeRequest
 - returns the authenticator
 - ChallengeResponse
 - sends the username and encoded password, based on the authenticator and SecretKey.
 - TicketResponse
 - returns a valid ticket with the timeout.

The available interface of AuthorizationControl is AuthorizationControlProvider, see Figure 13.

The AuthorisationControlProvider interface provides a clearance for an specific system to consume an specific service. The method is

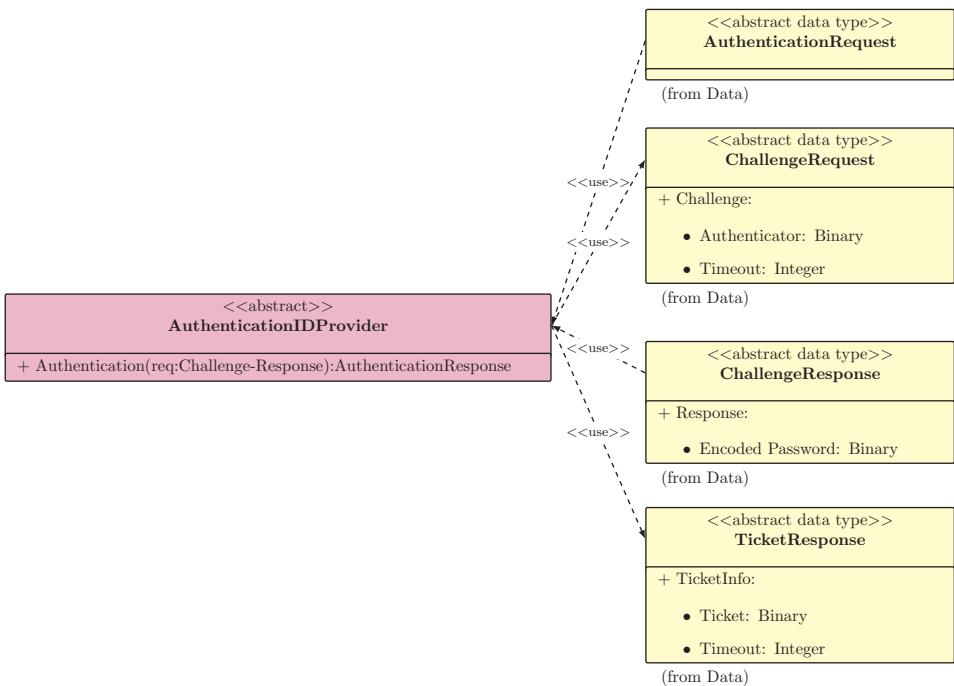


Figure 12: The interface definition and available data-types of the Authentication service.

- Authorise with the data types
 - **AuthorisationRequest**
request a validation of an specific ticket from an specific remote consumer.
 - **AuthorisationResponse**
returns the validity of a specific ticket, returning extra information about the owner of that ticket.

The available interface of the AuthorizationManagement service is `AuthorizationManagementProvider`; see Figure 14.

The `AuthorizationManagementProvider` interface provides a number of methods to manage authorization policies

- Authorise with the data types
 - `ListPolicies()`:
to list all available policies.
 - `AddPolicy(policy):`
to add a specific policy.

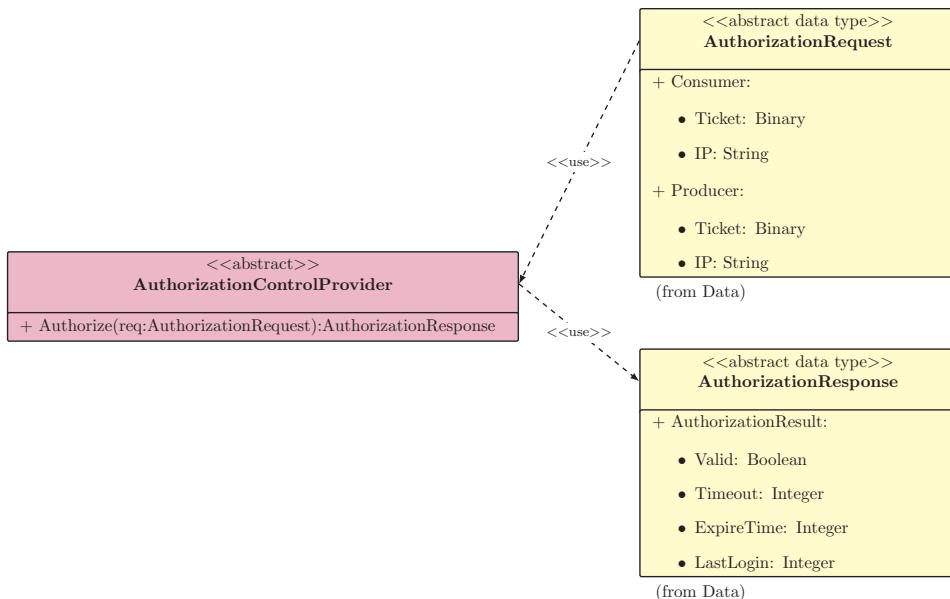


Figure 13: The interface definition and available data types of the *AuthorisationControl* service.

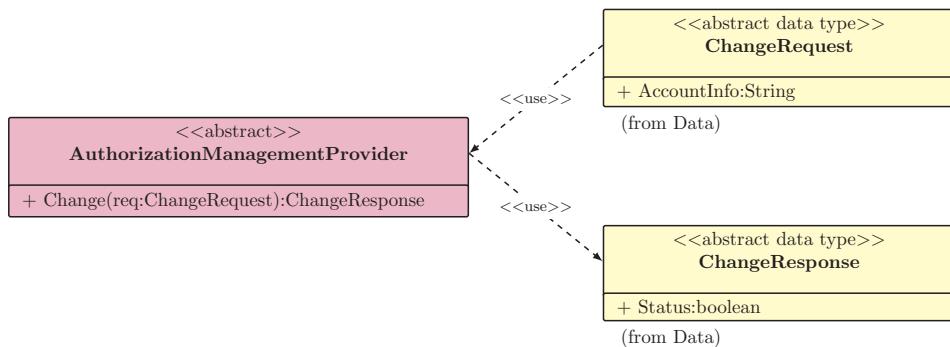


Figure 14: The interface definition and available data types of the *AuthorisationManagement* service.

- `RemovePolicy(policy)`:
to remove a specific policy.
- `ModifyPolicy(policy,new_policy)`:
to modify an specific policy.
- `ListAccounts()`:

to list all accounts.

- AddAccount(accountInfo):
to create a new account.
- RemoveAccount(accountInfo):
to remove a specific account.
- ModifyAccount(accountInfo,new_accountInfo):
to modify an specific account.

The most recent information on the AAA-Authorisation system is available in the Arrowhead Framework wiki [9].

MMI-Authorisation system

Within the Arrowhead Framework an MMI-Authorisation system has been defined and implemented to support operator interaction with the Authorisation systems.

The MMI-Authorisation system provides a graphical user interface that allows a user-/operator to manage and create access control rules for service producers.

The MMI-Authorisation system use the service AuthorisationManagement to communicate with the Authorisation system. The ServiceDiscovery service is used to list possible service for which access rules can be set.

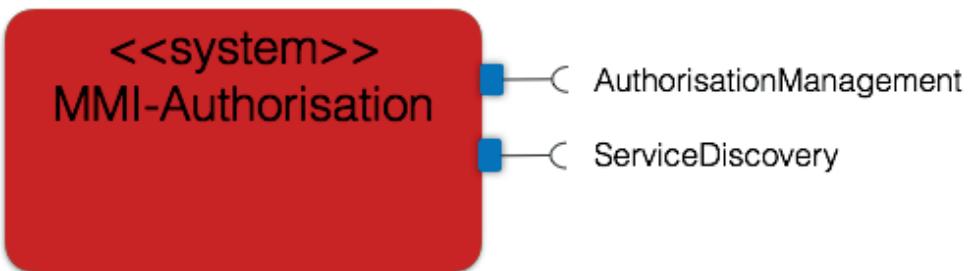


Figure 15: The MMI-Authorisation system and the services it shall consumed.

The MMI-Authorisation system do consume the AuthorisationManagement and ServiceDiscovery services, cf. Figure 15.

2.3 Orchestration system

The Arrowhead Framework implementation of the Orchestration system provides both the OrchestrationStore service and the Orchestration service.

The Orchestration system stores orchestration rules and resulting orchestration patterns. The requirements are specified during the design phase using engineering tools.

These are provided to Orchestration system either via the PlantDescription service or directly from the engineering tools. New requirements can be provided in runtime upon which the Orchestration system computes new orchestration patterns which are provided to the involved application systems.

Orchestration services

The Arrowhead Framework implementation of the Orchestration system produces four services OrchestrationStore, OrchestrationPush, OrchestrationCapability and OrchestrationManagement. The Orchestration system also consumes services of type `_ahfc-servprod`. There are currently two automation support core service of this type, the EventHandler service; see Section 3.4 and the Translation service; see Section 3.8.

OrchestrationStore service The OrchestrationStore service provides functionality for storing and retrieving orchestration requirements. An orchestration requirement is a set of rules which describe the ideal service required by a consuming system. This could be as simple as a fully defined service contract. But it could also be as complex as describing service instance requirements, such as physical or geographic qualities.

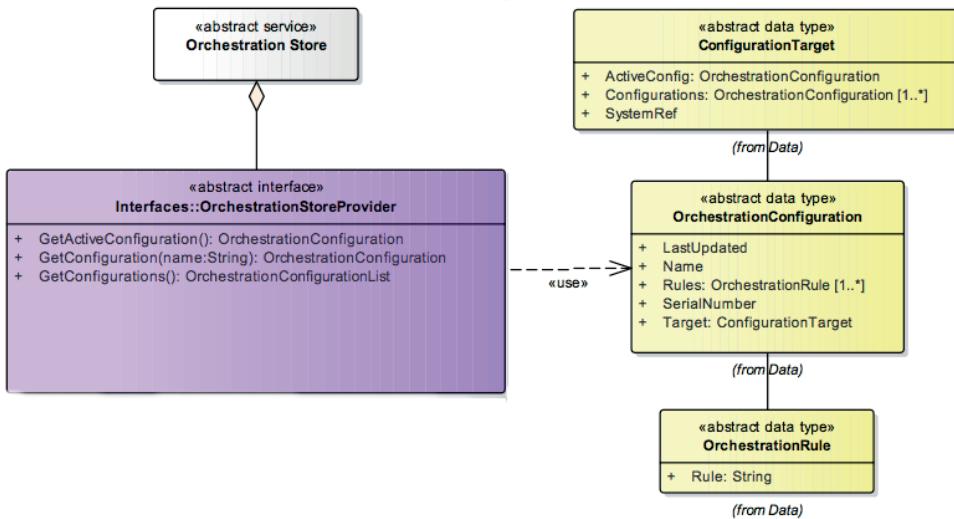


Figure 16: OrchestrationStore interface with methods and associated data definitions.

The OrchestrationStore interface with methods and associated data definitions is shown in Figure 16. The methods and associated data definitions are

- `GetActiveConfiguration` method is used to retrieve the currently active configura-

tion. That means the configuration that should be executed instantly.

- GetConfiguration method is used to get a specific configuration based on the name of the configuration.
- GetConfigurations method is used to retrieve all configurations for any system, i.e., the orchestration configurations for the whole System of Systems.
- ConfigurationTarget data type defines a system that can be orchestrated.
- OrchestrationConfiguration data type defines how a system should be configured via a set of orchestration rules.
- OrchestrationRule datatype defines a connection between a producer and a consumer. OrchestrationRule contains end point information and consumption pattern enabling.

OrchestrationPush service The Orchestration Push service is used to push orchestration configuration to an application system.

The consumer of the service has orchestration rules that describe a desired "end-state". The producer of the service is a system that has application services that should be connected to producers, in the Arrowhead network. By pushing the configurations from the consumer to the producer the receiving system will be able to connect its application service consumers.

The OrchestrationPush interface with methods and associated data definitions is shown in Figure 17. The methods and associated data definitions are

- PushOrchestration method is used to send orchestration configurations to the system that produces the service.
- ConfigurationTarget data type defines a system that can be orchestrated.
- OrchestrationConfiguration data type defines how a system should be configured via a set of orchestration rules.
- OrchestrationRule defines a connection between a producer and a consumer. OrchestrationRule contains end point information and consumption pattern.

The message sequences for PushOrchestration method is provided in Figure 18.

OrchestrationCapability service Orchestration Capability service provides the capability to extract the current state in which service is provided by which system and which system is consuming which service enabling possibility to relate systems and their services. The OrchestrationCapability can thus be used to extract the current state of the enabled service exchanges. The state information should be extracted from the OrchestrationStore. This provided that all systems have released the orchestrated right to the Orchestration system.

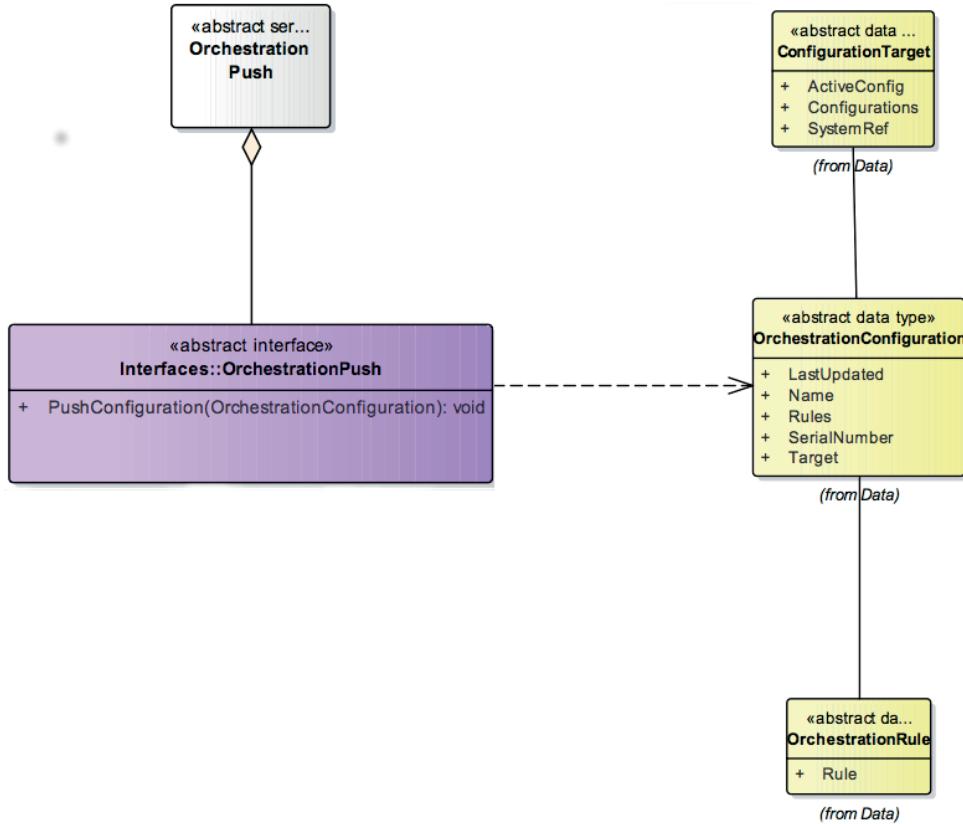


Figure 17: *OrchestrationPush* interface with methods and associated data definitions.

Provided that such state detection can be triggered simultaneously across all local clouds at a production site, essential state information and topology information can be gathered through the `OrchestrationCapacity` service.

The `OrchestrationCapacity` interface with methods and associated data definitions is shown in Figure 19. The methods and associated data definitions are:

- `GetCapacity` method returns a listing of available connections a service supports as well as the number currently connected to the service.
- `GetConsumerTypes` method returns a list of consumer types that the system will support.
- `GetProducerTypes` method returns a list of producer types that the system will be able to produce.

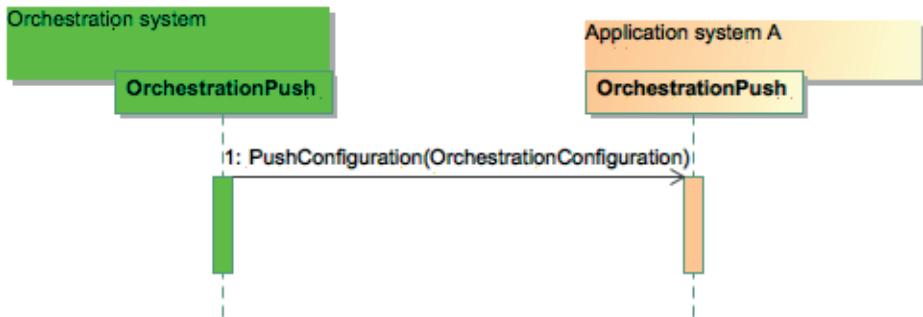


Figure 18: Sequence diagram for pushing orchestration configurations to an application system.

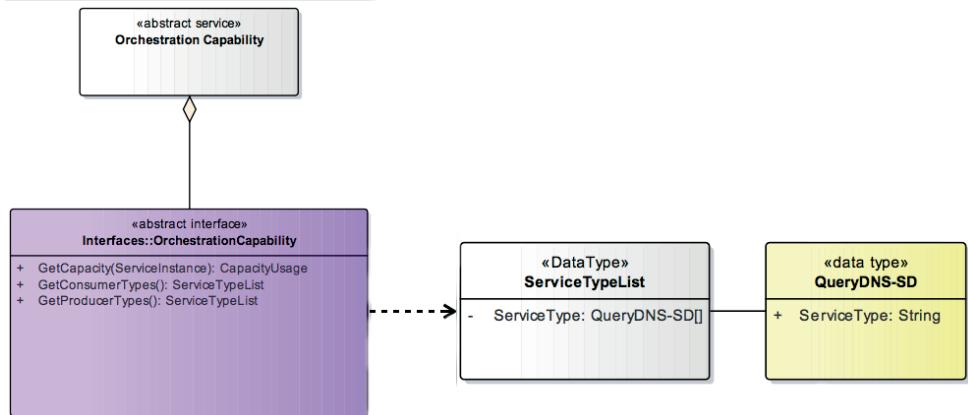


Figure 19: `OrchestrationCapability` interface with methods and associated data definitions.

- `QueryDNS-SD` data type implements the `Query` data structure using DNS address lookup. `ServiceType` is a string identifying the service name.

OrchestrationManagement service The `OrchestrationManagement` service handles management of orchestration configurations, including creation, editing, and removal of these configurations.

The `OrchestrationManagement` interface with methods and associated data definitions is shown in Figure 20. The methods and associated data definitions are

- `CreateConfiguration` method is used to create orchestration configurations
- `DeleteConfiguration` method is used to remove configurations for a specific target

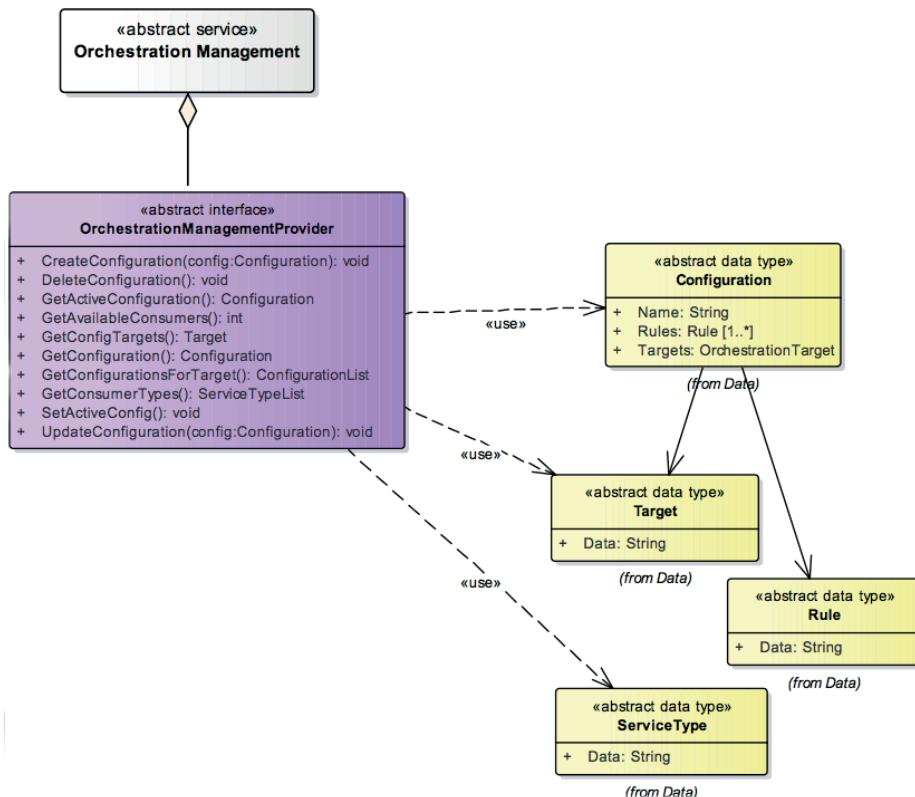


Figure 20: *OrchestrationManagement* interface with methods and associated data definitions.

- GetActiveConfiguration method is used to retrieve the currently used orchestration configuration
- GetAvailableConsumers method is used to determine the number of consumers of a specific type the system can serve simultaneously
- GetConfigTargets method is used to get a listing of the systems that can be orchestrated
- GetConfiguration method is used to fetch current orchestration configuration for the specific system
- GetConfigurationsForTarget method is used to fetch all orchestration configurations for a specific target
- GetConsumerTypes method is used to list the supported service types a system can consume

- SetActiveConfig method sets which orchestration configurations that are to be used
- UpdateConfig method is used to update a specific orchestration configuration
- Configuration datatype defines how a system should be configured via a set of orchestration rules.
- Rule datatype defines a connection between a consumer and a producer
- ServiceType datatype defines the type of service that a system provides Ideally this data type should be retrieved from the service registry service
- Target datatype defines the orchestration target, that is a system that should be affected by the orchestration rule(s)

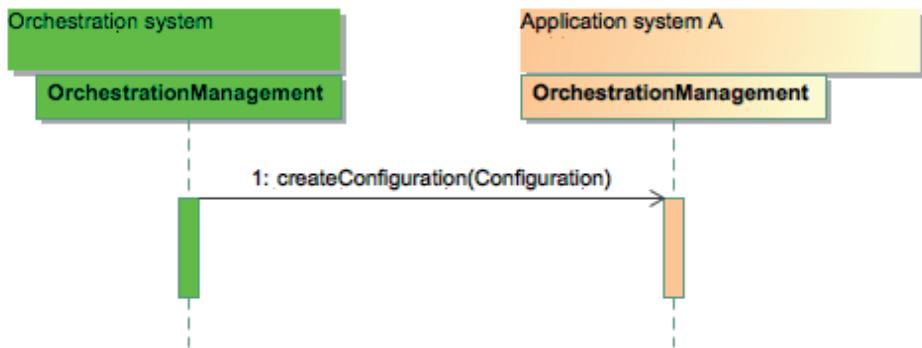


Figure 21: Sequence diagram for the dynamic behaviour when creating orchestration configurations.

The message sequences for all these interfaces is straight forward. Thus only the sequence diagram for CreateConfiguration is provided; cf. Figure 21.

MMI-Orchestration system

To manage the Orchestration system a GUI is provided through the optional MMI-Orchestration system. It provides a graphical user interface that allows an orchestration manager to create connection rules (orchestration) for systems (i.e., for system A and system B). The MMI-Orchestration system consumes two services, cf Figure22:

- ServiceDiscovery — to register itself into the local cloud
- OrchestrationManagement — where an orchestration manager can create or change an orchestration rule.

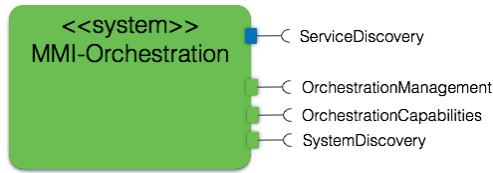


Figure 22: The MMI-Orchestration system and its consumed and produced services.

It is obvious that complex orchestration rules have to be generated by MES tools or similar. For now this is beyond the scope of this book.

3 Automation support core systems

To support the Arrowhead Framework architecture a number of automation support systems and associated services have been defined. Implementations of several of these are already available at the Arrowhead Framework wiki [9]. These systems and servies are described in detail below.

3.1 PlantDescription system

The purpose of the PlantDescription system is to provide a basic common understanding of the layout of a plant or site, providing possibilities for actors with different interests and viewpoints to access their view of the same dataset. These datasets are to be provided by the databases integrated in or generated by the flora of computer-aided design (CAD) tools used by engineers from all disciplines involved in designing a large automation System of Systems. More detailed engineering scenarios, including intended usage of the PlantDescription system, can be found in Section 2. It is clear that one plant description can span several local automation clouds.

A source of inspiration for the design was the standard ISO/IEC 81346 [10], which specifies that for studying objects and their relations it may be useful to look at them from different viewpoints, highlighting different aspects of the objects and relations. This standard is focused on the three aspects function, product, and location, although the design is intended to be capable of addressing other viewpoints as well.

In Figure 23 different types of information about a plant are illustrated.

- Blue represents the fairly static structure of a plant, such as locations, mechanical hardware, process equipment, or expected overall functionality.
 - This is what is typically provided by the plant description.
- Green represents the Arrowhead systems that are operating within the plant.

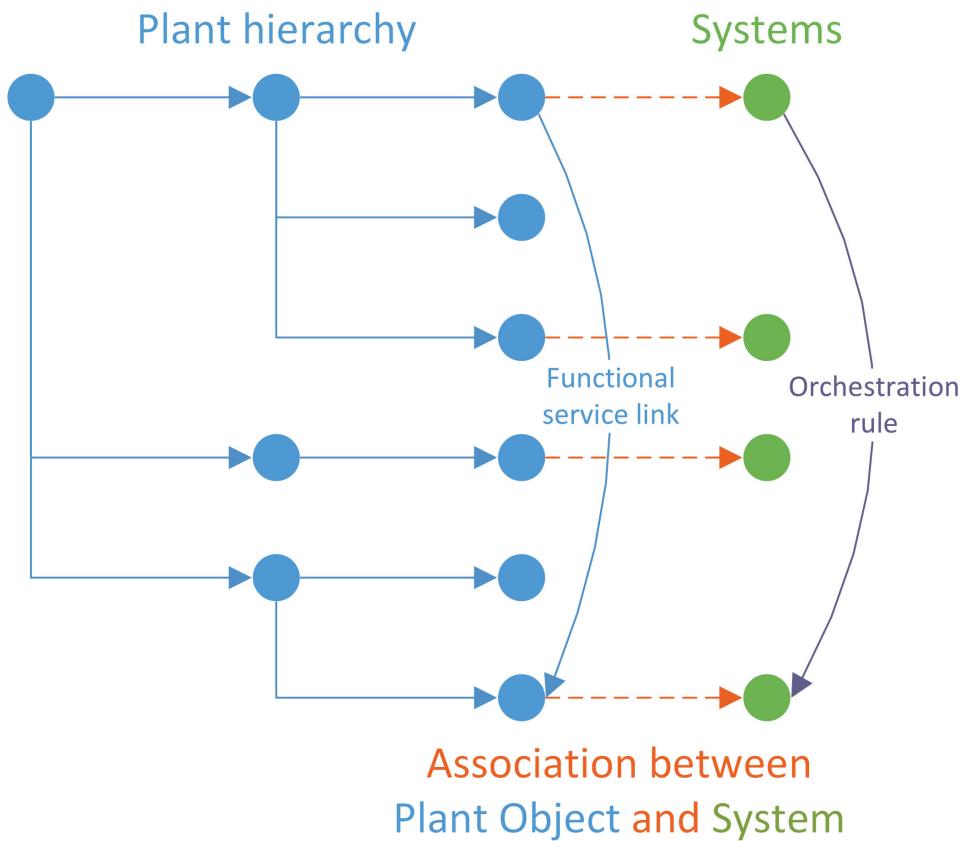


Figure 23: Distinctions between system responsibilities. Only the blue part of this diagram should typically be provided by the plant description.

- Information about the systems is typically stored as metadata within the systems themselves but also in the ServiceRegistry, the SystemRegistry, and the DeviceRegistry.
- Orange illustrates the link between the overall structure of the plant and the systems, and thereby shows how a system fits into the larger scale purpose.
 - These links may be stored by the systems themselves, in a configuration system, and/or as metadata in the SystemRegistry. For replacement purposes it may be useful to have the information stored outside of the system itself.
- Purple represents an orchestration rule, used to direct Arrowhead Framework sys-

tems to consume services of other Arrowhead Framework systems.

- Orchestration rules are generated by the PlantDescription system and provided to the Orchestration system.

Further details on the PlantDescription architectural and technology details can be found in [11, 12].

PlantDescription services

The two functions GetViewpoint and GetObjectsByNode are the main points of access to the PlantDescription system. An optional function GetObjectsByTypeId can be implemented to be used as a translator from one viewpoint to another.

GetViewpoint(Type): PlantData The function GetViewpoint is used with an argument (Type) to get all nodes and links that correspond to this type.

GetObjectsByNode(NodeId): PlantData The function GetObjectsByNode is used with a specific NodeId to get all nodes linked to the specified node, and those links.

GetObjectsByTypeId(Type, NodeTypeId): PlantData The function GetObjectsByTypeId is used with a specific combination of a Type and a NodeTypeId corresponding to the identity by which the node is identified according to that type. The function returns the same information as the GetObjectsByNode for that node.

Abstract information model

The information provided by the Plant description service consists of sets of nodes and links that can be used to describe different topologies, hierarchies, and structures for a large facility. Figure 24 describes the attributes of the three data types PlantData, Node, and Link.

The PlantData, Node, and Link datatypes are provided in Tables 1 2 and 3. In Table 4 a set of service metadata is provided.

Table 1: PlantData data type description

Field	Description
Nodes	A collection of (zero or more) nodes
Links	A collection of (zero or more) links

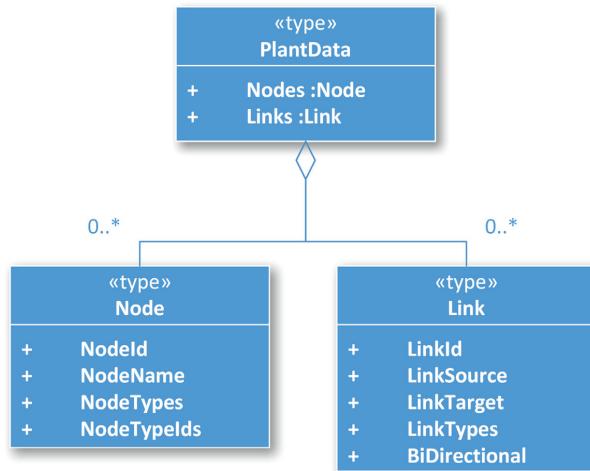


Figure 24: PlantData structure.

Table 2: Node data type description

Field	Description
NodeID	A name or description for the node that is useful for human interaction (optional)
NodeTypes	A collection of (zero or more) node types
NodeTypelds	An identifier or name specific for each type that the node belongs to (optional)

Table 3: Link:Data data type description

Field	Description
LinkID	A unique identifier for the link
LinkSource	The NodeID of the source node
LinkTarget	The NodeID of the target node
LinkTypes	A collection of (zero or more) link types
BiDirectional	Parameter describing if the link is bi-directional

3.2 Configuration system

The purpose of the Configuration system is to provide a uniform way for Arrowhead compliant system to manage distribution of configurations. The extent of the configurability of a system ultimately depends on the system itself and therefore the design of this service is intended to allow different levels of configurations to be transferred using the same interface, from changes in system parameters to full firmware updates that may change which services a system is able to produce and consume and other fundamental

Table 4: Service metadata description

Field	Description	Mandatory
Plant	Identifies the plant, site, area, network, or similar that the instance contains information on	Yes
ViewpointType	List of Viewpoint types supported by the instance	No
LinkTarget	The NodeId of the target node	Yes

properties.

Through this design the decision of how configurable a device or software system should be is left to the provider, and not imposed by the framework, while still allowing a uniform method for configuration management across diverse Systems of Systems containing devices and software systems with different levels of configurability.

The focus of the initial design of the Configuration system has been on the interface towards application systems, the service ConfigurationStore, indicating that further development of the service ConfigurationManagement may be desired once more partners take an interest in developing powerful management tools for the management of Arrowhead systems.

Configuration services

To provide the functionality included in this design the Configuration system provides two services, the ConfigurationStore and ConfigurationManagement. The ConfigurationManagement service is intended to be used by a user or other system to assign configuration files to configurable devices while the ConfigurationStore is the service to be consumed by systems on configurable devices to access the configuration files that have been assigned to them.

Depending on the which automation support core systems are available, what capabilities the configurable devices/systems have, and what security and identification mechanisms are used, the interaction scheme might look slightly differently. Figure 25 illustrates how the interaction might look in a scenario where the PlantDescription system is used to track which physical device is used to implement a specific function in the larger System of Systems. Here the device identifies itself using its serial number, and there are no explicit authentication and authorization interactions included. In a different implementation the identity used for interaction with the ConfigurationStore service may be provided to the hardware device or hosted software system during an initial deployment process, either upon request or as part of an earlier registration with the Authorization system or an DeviceRegistry system.

A special usage of the ConfigurationStore service is as a deployment mechanism for software systems, as discussed in Section 4.4. The authorisation of the ConfigurationStore service to configure a device is provided by the local cloud Authorisation system. In this way ConfigurationStore is supporting bootstrapping of Arrowhead Framework compliant

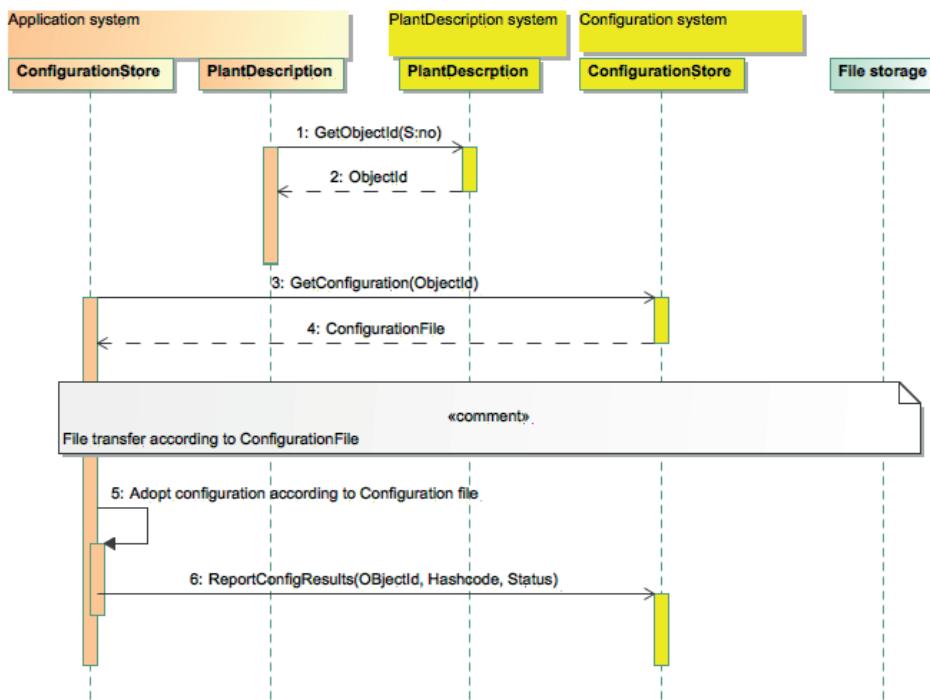


Figure 25: Sequence diagram for configuration of a device through the PlantDescription and Configuration systems.

services to a trusted device within a local cloud.

3.3 SystemRegistry and DeviceRegistry systems

Both the SystemRegistry and DeviceRegistry systems are in most aspect, a carbon copy of the ServiceRegistry system. The main and only difference are that they shall register and store which systems and devices currently are registered with the local cloud.

Currently no implementations are available of these two systems. Such implementation should, however, be straightforward to implement based on the ServiceRegistry system and the usage of DNS and DNS-SD technologies.

The SystemRegistry system is capable of storing reference information to all active systems and their produced services and their hosting devices. Consequently, the DeviceRegistry system is capable of storing reference information to all active devices and their hosted systems.

SystemDiscovery and DeviceDiscovery services

The SystemDiscovery and DeviceDiscovery services interfaces are defined according to Figure 26.

The service interfaces provide three methods:

- Publish

The publish method is used to register systems or devices. The services will contain a symbolic name as well as a physical endpoint. The instance parameter represents the endpoint information that should be registered.

- Unpublish

The unpublish method is used to unregister a service that no longer should be used. The instance parameter contains information necessary to find the service to be removed.

- Lookup

The lookup method is used to find and translate a symbolic service name into a physical endpoint, IP address, and a port. The query parameter is used to request a subset of all the registered services fulfilling the demand of the requesting system.

The returned listing contains service endpoints that fulfil the query.

The lookup, publish, and unpublish method sequences are provided in Figures 27, 28, and 29. The sequence diagrams for the DeviceDiscovery methods are almost identical.

The information model for the publish method holds two data types:

- SystemRecord

with the following data:

- Endpoint - string

This data type implements a representation of an endpoint using SRV record of DNS.

- * Hostname is a string containing the name of the host in format:

name.domain.topdomain, e.g., *app.arrowhead.eu*

- * path: 192.168.1.20

- * Port is an Integer containing the port number, e.g., 8070

- * Edata is a String containing additional information related to the endpoint. Any additional information that is required to identify the service instance should be stored in the mandatory DNS TXT record, as discussed in Section 2.6 and defined in [3]

- * Metadata - string

Metadata for the service are stored as key value pairs, e.g., time to live, sleep period, configurations, payload encoding, compression, and semantics. To allow for the orchestration to understand if any translation is necessary, it is proposed that the following three be mandatory:

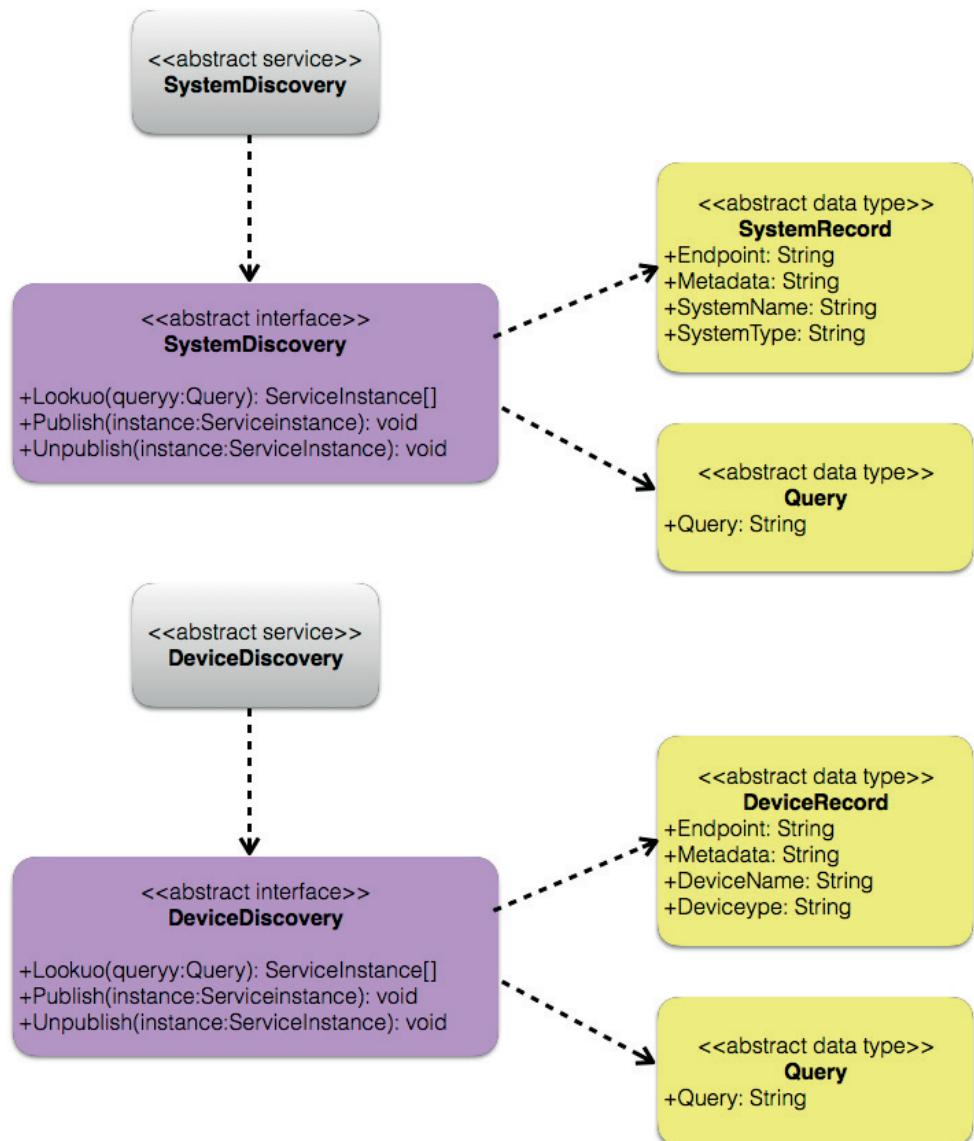


Figure 26: SystemDiscovery and DeviceDiscovery services with their three interfaces and associated SystemRecord and DeviceRecord data types and Query data type.

- `_encoding` is, e.g., `_xml` where XML [6] encoding is used and specified in the CP (Communication Profile) document

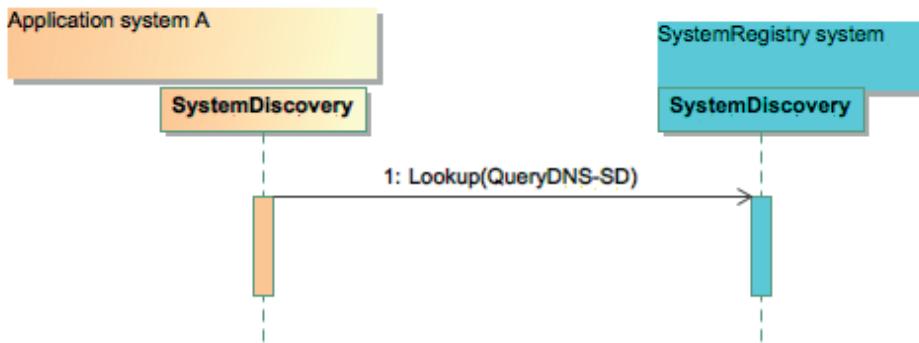


Figure 27: Sequence diagram for the lookup method of the SystemDiscovery service.

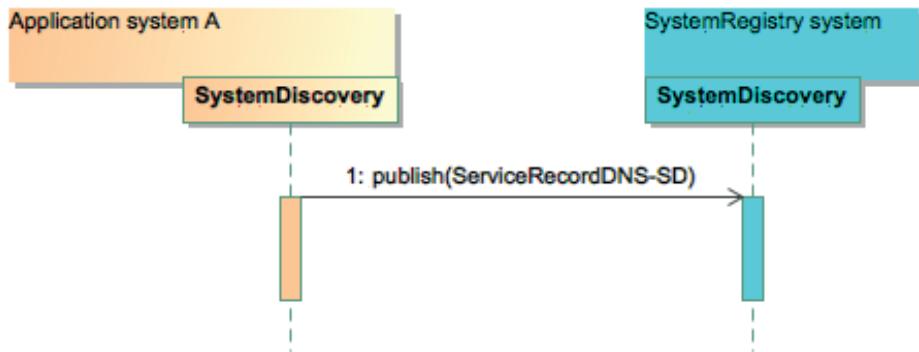


Figure 28: Sequence diagram for the publish method of the SystemDiscovery service.

- *_compression* is, e.g., *_exi* where EXI [7] compression is used and specified in the CP document
- *_semantics* is, e.g., *_senml* where SenML [8] semantics is used and specified in the SP (Semantic Profile) document
- * ServiceName - string
Name of the service instance, e.g., *_Temp1*.
- * ServiceType - string
e.g., *_ahf-temperature._coap._udp*.
- Query
with the following content:

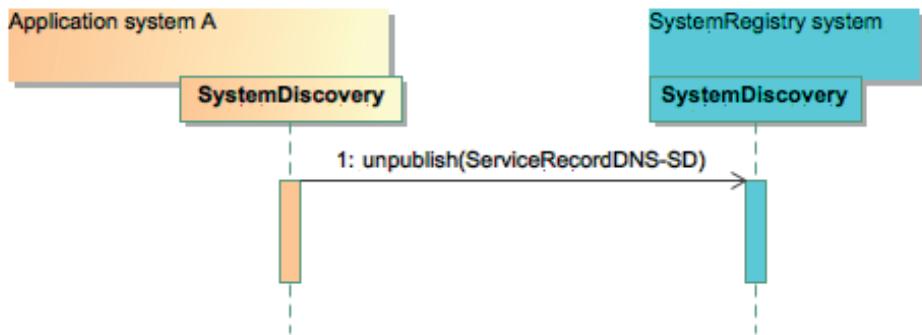


Figure 29: Sequence diagram for the unpublish method of the SystemDiscovery service.

- Query - string
where the query string specifies one or several of the data types of the service.
The query will then return a list of all registered services with the specified data type/s.

To access the ServiceRegistry from a REST based system Arrowhead Framework has implemented the ServiceRegistryBridge system allowing easy ServiceRegistry interaction from a REST—http protocol based system.

The sequence diagram for REST based system interactions with the ServiceRegistry is shown in Figure 30.

3.4 EventHandler system

The EventHandler core system is in charge of the notification of events that occur in an Arrowhead-compliant installation [13]. The Eventhandler system is available to support scenarios like:

- Providing resource and functional constrained service producers with publish subscribe capability
- Reduce service consumption load on resource-constrained producers
- Provide data filtering
- Support for QoS mitigation handling

EventHandler services

The EventHandler system produces the EventHandling service of the type *.ahfc-servprod*. The Orchestration system is the expected consumer of this service. Here a service request

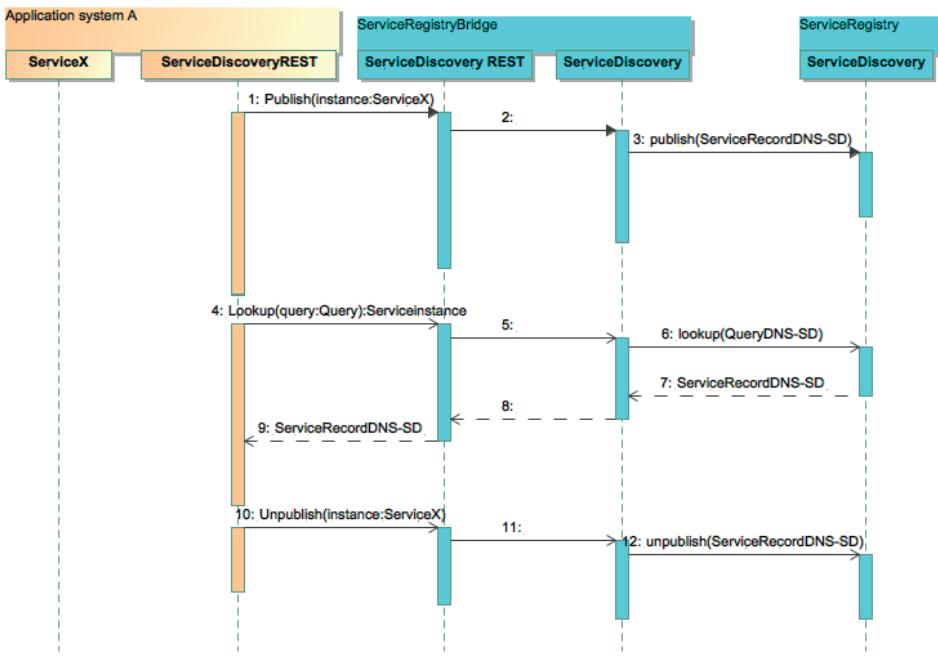


Figure 30: Sequence diagram of a REST based system A registration process with the ServiceRegistry as supported by the ServiceRegistryBridge.

with a specific event handling rule will trigger the EventHandler to create a transient instance capable of consuming a specific service and producing one service. The services to consume shall be specified in the event handling rule. The consumed service payload will be filtered according to one or several filtering algorithms specified in the event handling rule. The filtered payload will then be made available by the Eventhandler system as a transient produced service. This produced service shall support a publish subscribe capability. The created service shall be published to the ServiceRegistry. Filtering capabilities shall be provided as service meta-data in the form of key pairs in the ServiceRecord; see Section 2.1.

In addition, event handling rules may specify that the consumed service data should be stored to a Historian system by a transient consumer consuming the HDataIn service; see Section 3.6.

One or several consumers can then be orchestrated to consume such service produced by the EventHandler system. For a publish subscribe usage scenario, the Eventhandler will select the data filtering algorithm matching the subscription.

Figure 31 shows a sequence diagram for an EventHandler interaction with the Orchestration system, application systems and a Historian system.

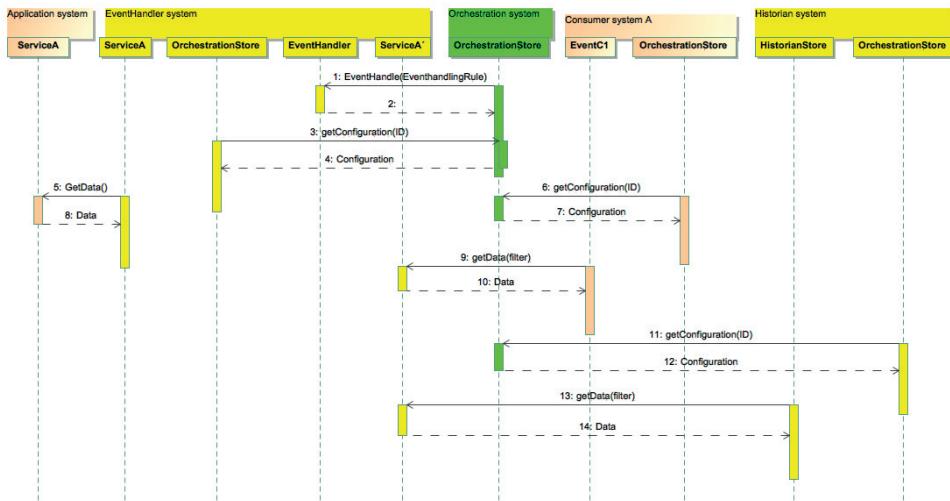


Figure 31: Sequence diagram for an EventHandler interaction with application systems and the Historian system as directed by the Orchestration system.

3.5 QoSManager system

QoS in a local cloud is managed through the QoSManager system [14], which produces the QoSSetup and the QoSMonitor services [15]. The QoSManager system is responsible for verifying the feasibility of QoS requests, configuring involved parties, and monitoring the performance of services to ensure that QoS requirements are satisfied. The QoSManager system is designed to work in close collaboration with the Orchestration system, and in most scenarios the Orchestration system is the only consumer of the QoSSetup service.

In a distributed setting, QoS capabilities require the involvement of not only the producer and consumer of services, but also network actives that are mediating data transfers in the system and the devices that are hosting the services. The QoS dimensions that can be tackled within this general setting are

- End-to-end **delay** - for hard/soft real-time guarantees;
- Message/service **prioritisation** — when no real-time guarantee is available;
- Data/message **bandwidth** and computational bandwidth — to accommodate enough service requests;
- Communication **semantics** - delivery guarantees and message ordering.

For each QoS dimension, a number of parameters are used to express QoS requirements quantitatively. Table 5 presents a list of the QoS parameters for each QoS dimension, together with their meaning.

End-to-end delay for a service invocation is a very common non-functional requirement in distributed automation System of Systems, and its QoS objective implies the execution of actions within a deadline. This class of objectives comprises both hard real-time and soft real-time constraints, the latter representing statistical guarantees on the communication delay.

Message/service prioritisation involves the ordering of communication flows and service requests, based on their urgency. When network actives and devices hosting Arrowhead systems are concerned with messages and operations related to a service, they will treat them as having higher or lower priority with respect to other applications based on these QoS parameters. For this purpose, network actives use priority queues to manage the messages, and the devices use priority scheduling algorithms in favour of more urgent services.

Data/message bandwidth refers to guaranteeing sufficient communication and computational resources for a service. This class of QoS requirements is quite common for service support in SOA applications and slightly less common in embedded applications. This class of objectives comprises both constraints on the minimum bandwidth for data produced/transmitted in a time unit, and on the number of service requests supported in the time unit. Usually, the QoS requirement for requests for bandwidth is applied to service producers only, since network actives do not track the number of requests and thus limit their vision over the data bandwidth being used. On the other hand, requests for data bandwidth are used on both network actives and service providers.

The communication semantics protect the service interactions against events disrupting the communication infrastructure and the System of Systems in general. This class spans a set of capabilities that can be requested as part of the QoS. In particular, this QoS class is used to request assurance of receiving the message at least once, the assurance of not receiving duplicate messages, and the reception of messages in the same order they were produced.

Some scenarios can guarantee only a subset of the QoS capabilities described above. In particular, delay objectives can be respected in local cloud scenarios, where the network actives are under control by the same stakeholder that owns the systems; on the other hand, if the service consumer and the service producer are connected by a wide area network such as the Internet, the in-between network is not under control, real-time objectives are not feasible, and either prioritisation can be provided by means of solutions similar to Diffserv [16], or communication must operate in best effort.

The setup of QoS is a process strongly correlated with the service orchestration, performed through the Orchestration service. Arrowhead services are consumed after they are orchestrated together and obtained through interaction with the Orchestration service of the Orchestration system, and this latter system is in charge of accessing the QoSSetup service of the QoSManager system, which takes care of verifying that the QoS objectives are feasible, of keeping track of resource reservation, and of configuring the network actives and the devices according to the QoS objectives. Thus, this paradigm implies that the service request sent to the Orchestration Systems contains both the functional requirements of the service and the QoS objectives, which are expressed by

Table 5: *QoS parameters*

QoS dimension	Parameter	Type	Description
Delay	hardRT	boolean	whether it is hard real-time or soft
	deadline	integer	maximum delay for the end-to-end service consumption, in milliseconds
Prioritisation	prio	integer	relative priority of the communication flow or service
Bandwidth	requests	integer	minimum number of service consumptions that must be satisfied per second
	data	integer	minimum number of megabytes that must be produced by the service or transferred by the network active
Semantics	atLeastOnce	boolean	at least one copy of the message must be received by its recipient
	atMostOnce	boolean	no more than one copy of the message can be received by its recipient
	ordered	boolean	messages in this communication flow must be received in the same order they were generated

means of a Service Level Agreement (SLA).

Regarding the monitoring of the QoS, the system offering the QoS Monitor service monitors the performance of services, either directly by having modules running over network actives and systems, or by accessing logs of network actives and systems, to detect if QoS parameters are not being guaranteed by the currently orchestrated service instance; in this latter case, the QoS Monitor service will contact either the Orchestrator system or the service consumer through the EventHandler system, to instruct them to repeat the orchestration process. Depending on the configuration at hand, the service can be offered by the QoS Manager system itself, or by a system positioned strategically in the System of Systems. In the latter case, the QoS Monitor service will be used by the

QoSManager system to configure its monitoring operations.

An example of exchange of messages related to the orchestration of a service offering QoS is given in Figure 32.

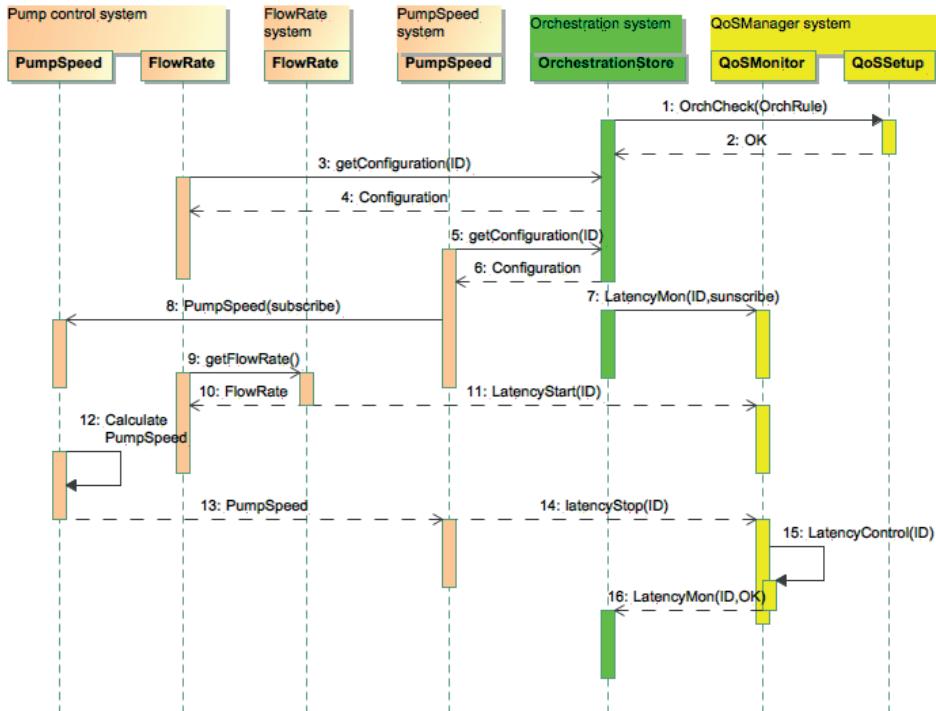


Figure 32: Sequence diagram for the orchestration of a simple control loop involving QoSSetup and QoSMonitor services.

QoSSetup service

The QoSSetup service is used to verify the feasibility of QoS objectives for a given orchestration. The information returned by the QoSSetup service influences the Orchestrator system to change device and/or system network and computational parameters to find a setting for which the QoS constraints can be met. The response to the Orchestration system is either QoS-OK or QoS-not-OK. On receiving a QoS-not-OK response, the Orchestration system has to provide a new orchestration proposal. It is possible that the proposed procedure may end up in an endless number of iterations, but the problem of the convergence of the QoS and Orchestration interaction is beyond the scope of this

book and a matter for further research.

The QoSManager shall be able to retrieve data from the ServiceRegistry, SystemRegistry, DeviceRegistry, OrchestrationStore, and ConfigurationStore within the local cloud. Based on data from these registries and stores, the QoSManager system will be able to deduce QoS constraints as well as available computational and network resources. The algorithms used for computing QoS feasibility and for smart QoS management are beyond the scope of this book.

QoSMonitor service

The QoSMonitor service performs online monitoring of the performance of the services, network actives, and devices hosting the systems, aiming at verifying whether QoS objectives are attained or not.

A simple latency monitoring process is visualised in Figure 32, where the QoSMonitor is orchestrated to subscribe to the service exchanges being part of time critical paths. The services exchange time-stamped data, enabling the QoSMonitor to calculate total latency from source to receiver.

It is clear that more advanced monitoring of QoS may be necessary within the local clouds of larger automation systems. Anyway, this topic is beyond the scope of this book and a matter for further research.

3.6 Historian system

The Historian system is used for storage, processing, and visualisation of data created by services in a local cloud, e.g., sensor data. Since the Historian system should be able to consume a large number of simultaneous active services, it should be executed on a sufficiently fast computer.

Since a Historian system may interact with a potentially very large number of heterogeneous devices and systems, it is beneficial for it to support as wide range of protocols and data models as possible.

The Historian system can be orchestrated to consume any service in the local cloud. The consumed data from a specific service will be stored in a file being named as the consumed service endpoint. Data stored are service payload plus metadata provided by the used protocol.

Historian service

The produced Historian service has one interface: PutData, with the data types shown in Figure 33. Every access to the PutData interface will create a file in the Historian with a filename of (service_endpoint.timestamp).

Figure 34 shows the procedure for a sensor platform that wants to store its sensor reading in a Historian system.

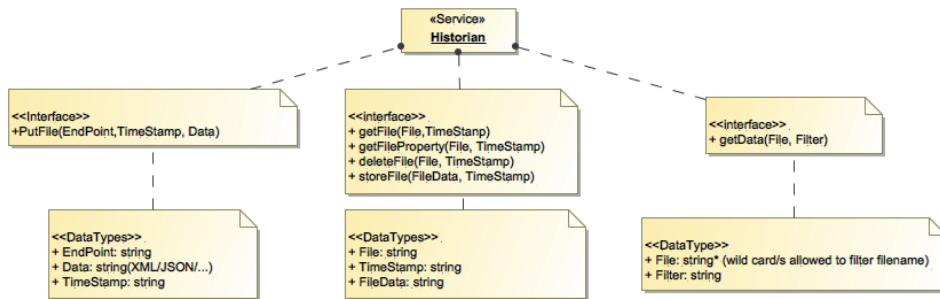


Figure 33: Historian interfaces and data types.

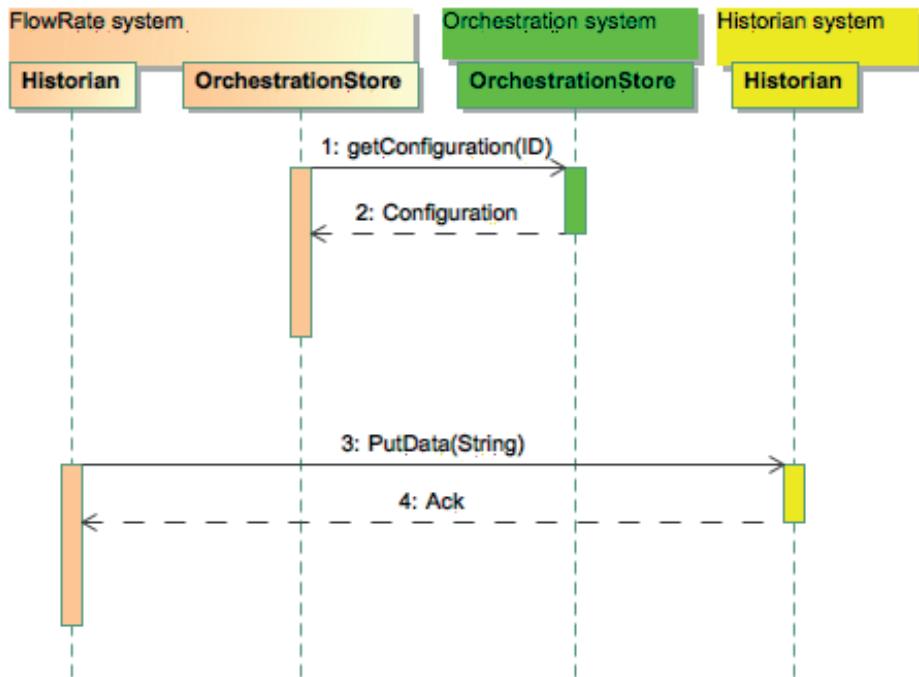


Figure 34: Application system interaction to store data in the Historian system data storage.

FileSys service

Service consumers, which can be human operators or other systems, that need to obtain data stored in the Historian system can use the FileSys service. The FileSys service has

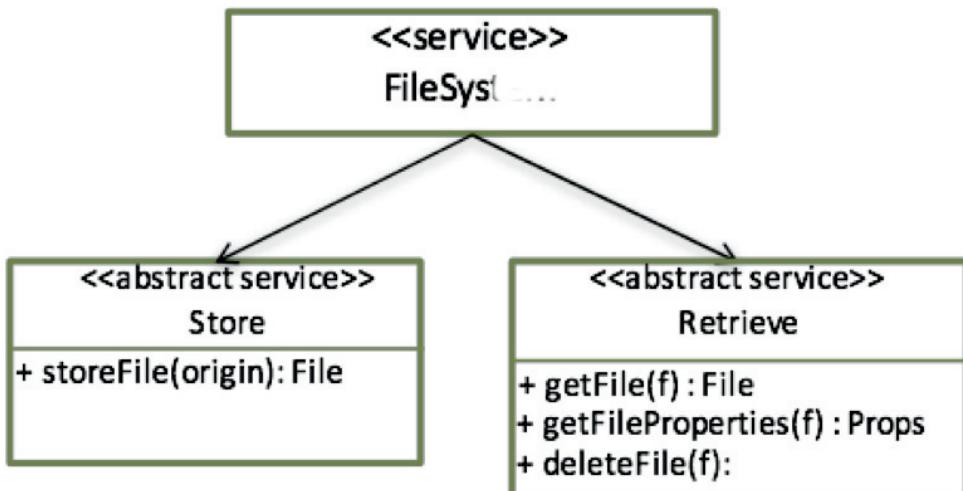


Figure 35: Interface definitions of the *FileSys* service.

interfaces and data types shown in to Figure 35.

Figure 36 shows the procedure for an application system that will retrieve and subsequently delete data from the Historian system.

This service enables clients to store and delete files and folders like a distributed file system. This service also supports generation of HTML data for creating web-based interfaces for all files and folders that are stored in a Historian system. This allows sensor data, or any type of files, to be stored, viewed, and downloaded using standard web browsers. The HTTP-based FileSystem service also enables third party software, e.g., MATLAB, Simulink, or Excel, to be used for data processing and visualisation.

Filter service

The Filter service enables application consumers to retrieve data from the Historian based on the content of the stored data.

For the latest development of the Historian system and its service please refere to the Arrowhead Framework wiki.

Service information data

The Historian's data model is primarily based on the SenML specification. Encoding is either JSON, CBOR, or XML. Other formats may be supported and can be queried at run time. The Historian understands and decodes all units currently defined in SenML and can take appropriate actions depending on filter rules, etc. This service can also be used to monitor when new systems, and existing ones for that matter, initiate data communication.

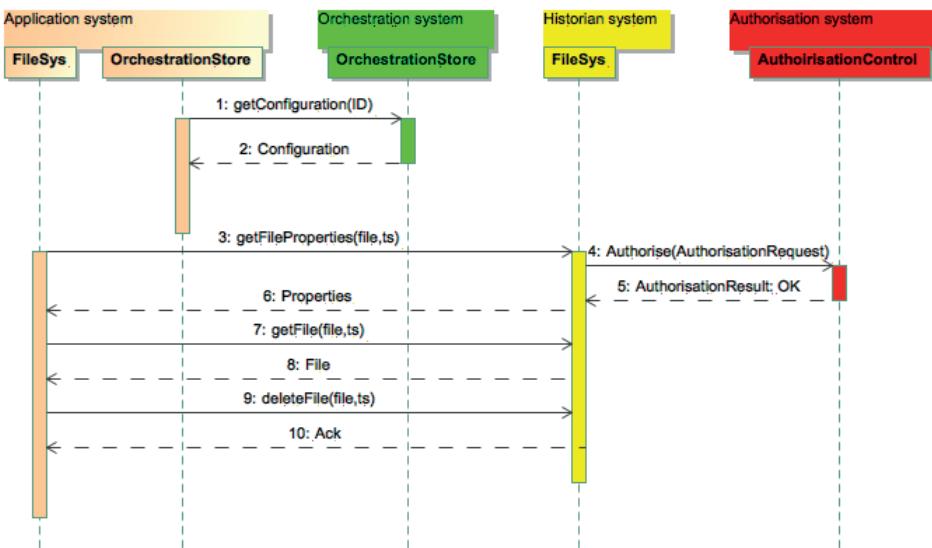


Figure 36: Sequence diagram for Historian system data retrieval and deletion.

Service metadata

The Historian also supports dynamic querying about active devices and systems. Clients can ask a Historian for all devices or systems, i.e., data sources, that have been active, for example, current day, last week, etc. Errors are represented using the SigML format. When using the file system feature, the Historian system is encoding and semantics agnostic.

The current implementation of the Historian's services supports a multitude of protocols, encodings, and data formats. For the latest specifications please consult the Arrowhead Framework wiki https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Support_Core_Systems_and_Services#Historian_system.

3.7 Gatekeeper system

There is a valid need for inter-cloud servicing, as one single Arrowhead cloud cannot serve all demands. Here are some examples (use cases) when inter-cloud relations are necessary or simply better than handling requests at a local level:

- Servicing is not possible: no service provider locally.
- Service provider is not available: is in out-of-order status or registered, but not found at the time.

- Servicing is currently not possible: no free resources or the QoS expectations cannot be met locally.
- Servicing within the home cloud is not optimal (e.g., geographically a different cloud is closer or better).

One of the major philosophical questions here is about data ownership, whether the command over servicing belongs to the system (e.g., it can choose its partners) or to the managing central entities of the local clouds (and these core systems pair up systems for servicing). As various intelligent decision-making processes (e.g., resource management) are implied here, in our primary scenario we assume that

- data ownership belongs to the local clouds (but this is not a necessity of the Gatekeeper services concept)
- one identity of an Arrowhead System of Systems cloud is requesting a service that cannot be fulfilled in the local cloud and therefore
- it might be viable to look for clients in other Arrowhead clouds

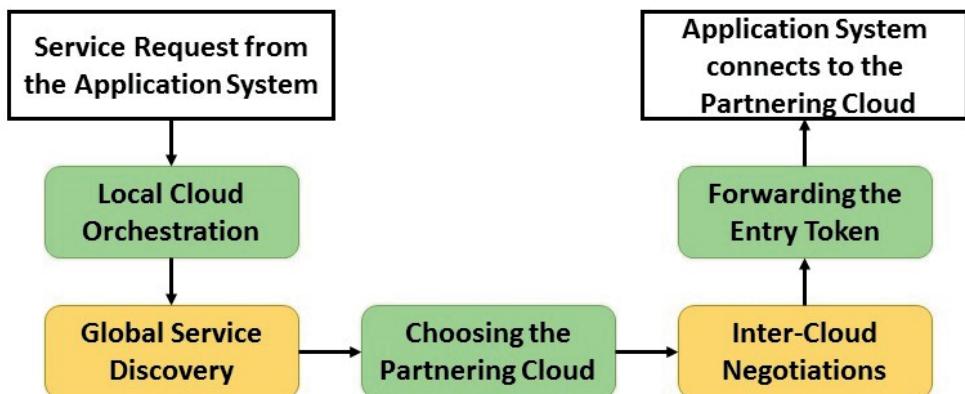


Figure 37: Creating inter-cloud consumer-producer relationship methodology.

The Gatekeeper system provides the following functionality (see Figure 37):

1. *Global service discovery:* Finding other Arrowhead local clouds with suitable providers for the requested purposes.
2. *Inter-cloud negotiations:* authentication, identity verification of the chosen partnering cloud, authorising transactions, and establishing secure connections by managing the dialogs (“negotiations”) between the clouds for establishing the inter-cloud producer-consumer relationship.

Global service discovery

There are several approaches to track the available services outside the local cloud. These approaches are suitable for different environments and requirements depending on the regularity of inter-cloud interactions, the volatility of service availability in the different clouds and the flexibility required from the collaborating Arrowhead System of Systems.

The most basic approach is hardwiring — and manually configuring — information about a certain set of other Gatekeepers into each Gatekeeper they can turn to; see Fig. 38. This can be based on the service type, time of day, geographical requirements (choosing a physically close partner), etc. This concept brings several advantages and limitations as well.

In this scenario, cloud operators have strict oversight of their Arrowhead clouds and can easily log the interactivity, set up billing, and access control between operators (fits business purposes). It is also highly secure: no need for authentication, identity control, or trust management in any phase. However, there is the need for manual editing of configuration and hence there is absolutely no option for scaling automatically.

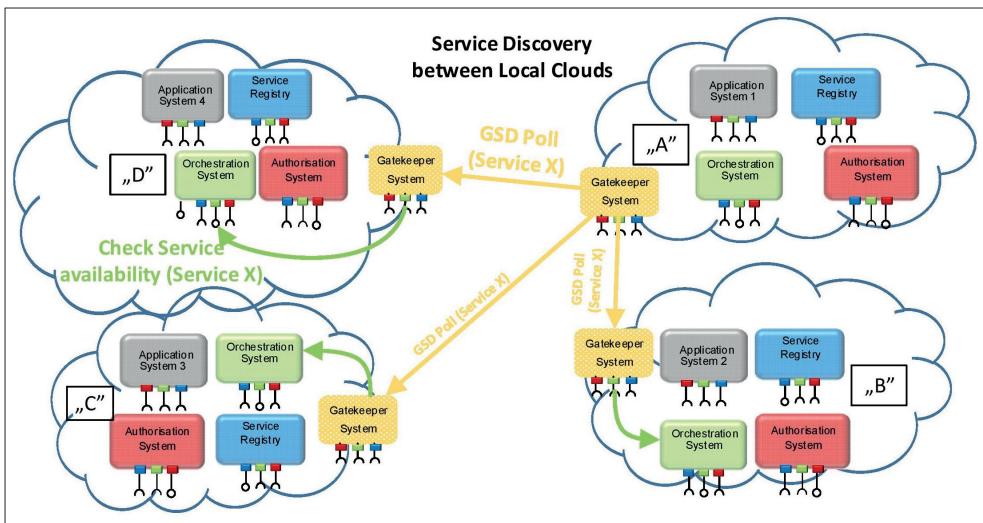


Figure 38: Service discovery in a hard-wired inter-cloud scenario.

The second approach is still based on per-transaction polling of neighbours, but lets the Gatekeepers automatically detect and poll their neighbourhood. The ad hoc peer-to-peer detection of neighbours can provide adaptivity for volatile environments (e.g., moving cars looking for charging stations along the road). However, it will also pose a high administrative overhead (for Gatekeeper components) and cause scaling issues. This methodology will also present a limited list of available resources to the local cloud

(just from the current contacts). There are several problems to solve here, such as the tracking, trust management, and authentication of a rapidly changing neighbourhood.

The third approach is pointing towards the creation of a dedicated inter-cloud system, where the “demand and supply” of inter-cloud requests can meet. The main purpose of this is to take off the computational overhead from the Arrowhead cloud Gatekeepers and local core systems in a stock-market, like environment; see Fig. 39. This centralised entity could provide global matchmaking for the participants of the Cloud of Clouds and could also centralize resource allocation and trust management. This can be achieved by tracking the reliability of the parties and providing the requester clouds with only the globally optimal partnering cloud.

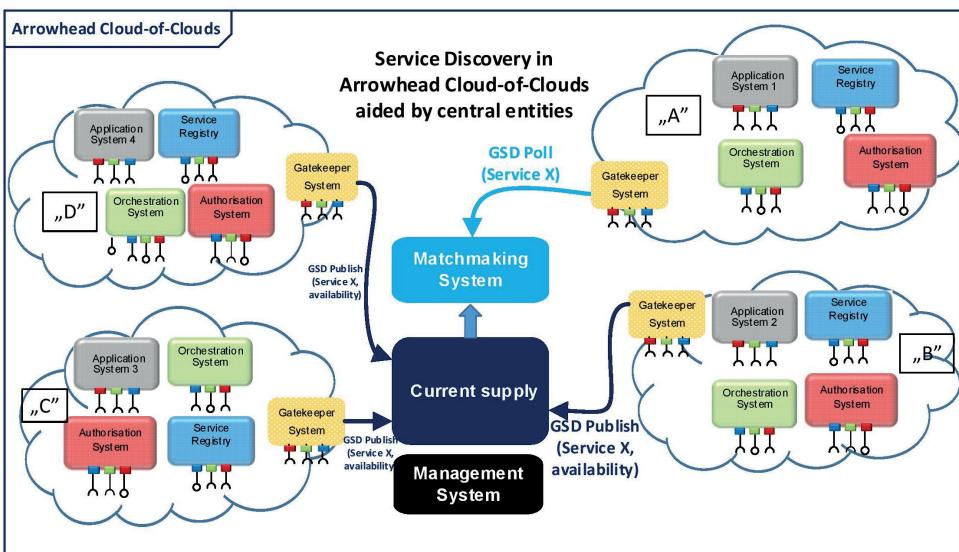


Figure 39: The Cloud of Clouds concept.

Inter-cloud negotiations

After the requesting cloud decides for a partnering cloud, the Gatekeepers of these clouds will have to perform negotiations to settle the different aspects of the transaction. At the end of this phase, the requesting system gets a token that will authenticate and navigate it in the partnering cloud, using the local ServiceDiscovery service. During this process, the following issues are to be handled:

1. protocol negotiations (Arrowhead Framework version and protocol matching)
2. mutual authentication and identity checking of the parties

3. actual admission control (authorisation and resource allocation for the transaction in the clouds)
4. establishing a secure connection between the Gatekeepers, therefore the Arrowhead clouds (creating the data path)
5. sharing the local ServiceDiscovery data of the partnering cloud (addressing the partnering cloud from the outside)
6. injecting temporary authorization information into the Authorisation system in the partnering cloud (access control of the “foreigner” system)
7. the partnering cloud Gatekeeper issues a token for the requester with temporary entry and shares it with the requesting Gatekeeper
8. the requesting local cloud forwards this token to the requesting system,
9. the system connects to the producer using the partnering cloud’s core Systems

3.8 Translation system

The Translation system provides the transparency capability to Arrowhead local clouds. The Translation system hosts on-demand proxy capability as transient service endpoints. That is, the Translation system can be requested to mimic a particular service provider and consumer pair whose service contract does not match. This is shown in Figure 40.

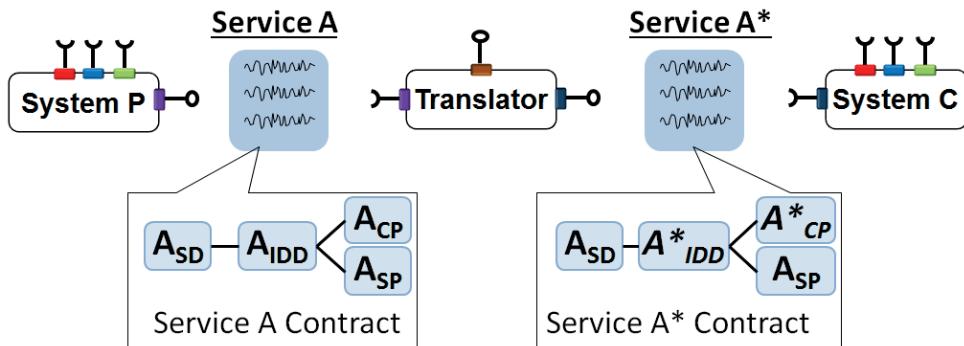


Figure 40: Arrowhead service contract mismatch.

As shown in Figure 41, the Translation system consumes the ServiceRegistry service and the Authorisation service and provides a Translation service. The interfaces used for actual translation are transient services and not registered to the service registry. Thus they are dedicated to the consuming systems making the request.

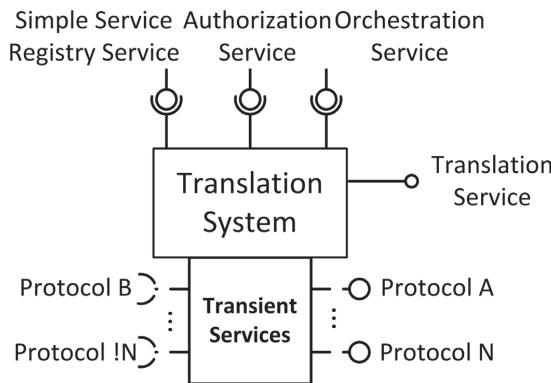


Figure 41: Translator service interface architecture.

The translation system currently supports translation between HTTP, CoAP, and MQTT. Figure 42 shows the block diagram of the translator. Each protocol has been implemented as a spoke segment which connects to any other spoke segment. Through the use of an intermediary structure new protocols can be introduced through the implementation of a service provider spoke and a consumer spoke. Then, based on the protocol translation requested, the hub will connect the appropriate protocol spokes. Thereby achieving efficiency of direct protocol-to-protocol translation while only requiring implementation of translation to/from the intermediary structure.

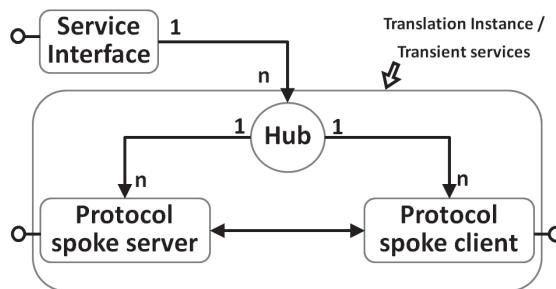


Figure 42: Translator block diagram.

The translation system architecture additionally allows for evolution of more advanced translation as each spoke can be cascaded as segments. With the addition of semantic, security, or encoding segments, it would be possible to fully support translation between domain-specific communication.

To overcome differences in protocol interaction patterns, a protocol spoke must 1) not rely on any behaviours from a paired spoke and 2) must take proactive behaviours

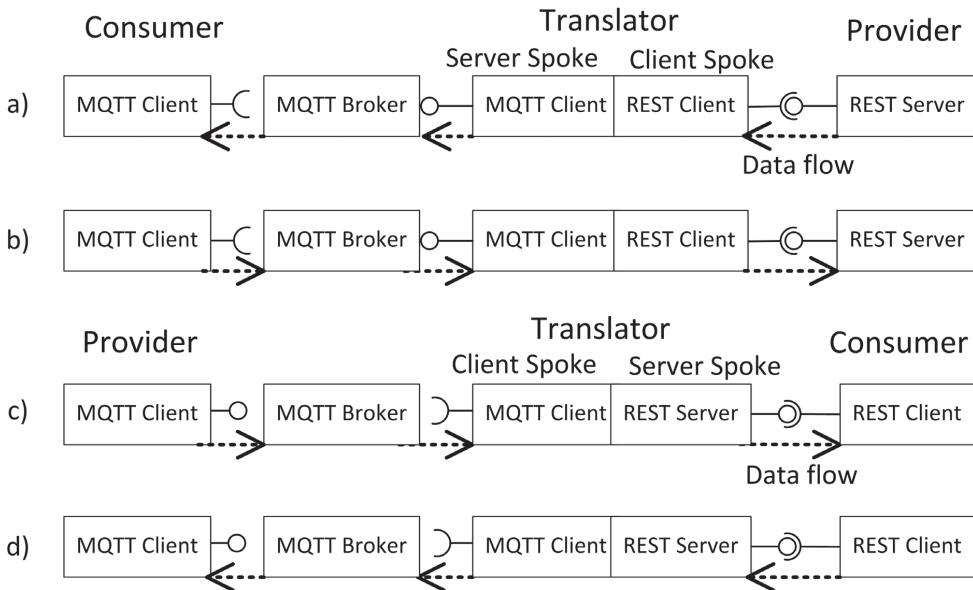


Figure 43: MQTT translator operational states.

depending on provider/consumer orientation. For example, in Figure 43, there are four operational modes for a translator bridging an MQTT spoke and a REST (CoAP/HTTP) spoke. This is bridging RESTful communication with a publisher-subscriber pattern. In Figure 43-a and 43-b the MQTT client needs to send/receive data to/from the REST server. In Figure 43-c and 43-d the REST client would like to send/receive data to/from the MQTT client.

The Translation system consumes the service registry and the authorisation service, and produces only a single discoverable service. The translation system dynamically creates service endpoints; however, these are not registered with the service registry as they are only available for use by specific systems which are notified directly by the Orchestration system.

Translation services

The Translation service is used to instantiate new translation instances. The interface definition can be seen in Figure 45. It takes the service provider service registry record and the service consumer type reference.

Based on this information the translation service will create a new translation hub which will host the protocol spokes. One spoke will make service invocations to the specified service provider and one spoke will mimic the service provider and await service requests from a service consumer. The service endpoint, HTTP/CoAP URL or MQTT

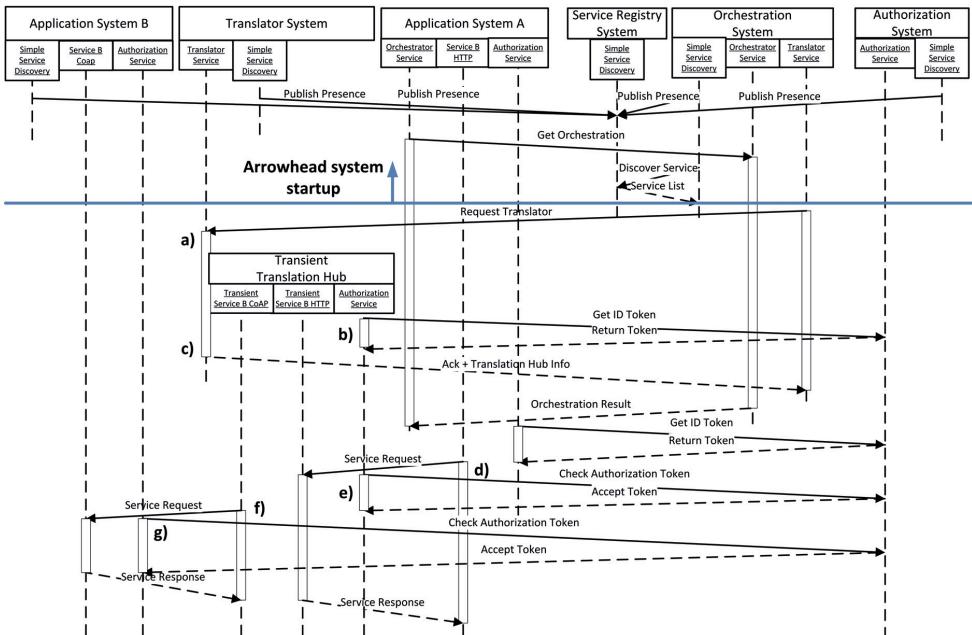


Figure 44: Translator sequence diagram.

broker+topic, will be returned to the service invoker. Each new request to the translator will create a new translation hub.

For the latest update on the Arrowhead Framework and the local automation cloud systems and services, please consult the Arrowhead Framework official wiki [9].

References

- [1] P. Mockapetris, “Domain names - concepts and facilities,” IETF, Tech. Rep., 1987.
- [2] ——, “Domain names - implementation and specification,” IETF, Tech. Rep., 1987.
- [3] R. Elz and R. Bush, “Clarifications to the DNS specification,” IETF, Tech. Rep., 1997.
- [4] A. Hubert and R. van Mook, “Measures for making DNS more resilient against forged answers,” IETF, Tech. Rep., 2009.
- [5] S. Cheshire and M. Krochmal, “DNS-based Service Discovery,” IETF, Tech. Rep., 2013.

```

<function name="get translator">
  <URI name="http://URI:PORT/translator">
    <method name="POST"/>
    <payload>
      <[Mandatory]providerName="string" />
      <[Mandatory]providerAddress="string" />
      <[Mandatory]providerType="string" />
      <[Mandatory]consumerName="string" />
      <[Mandatory]consumerType="string" />
      <[Optional] Event Service name="string" address="string" />
      <[Optional] Quality of Service record>
        < latency="string" />
        < throughput="string" />
        < jitter="string" />
        < packetloss="string" />
      </[Optional] Quality of Service record>
    </payload>
    <response>
      <Status Code 200>
        <[Mandatory]translatorId="int" />
        <[Mandatory]translatorAddress="string" />
      </Status Code 200>
      <Other Status Codes>
        <no payload in this response />
      </ Other Status Codes >
    </response>
  </URI >
</function>

```

Figure 45: Translator interface description

- [6] “Extensible Markup Language - XML,” 2016. [Online]. Available: <https://en.wikipedia.org/wiki/XML>
- [7] D. Peintner and S. Pericas-Geertsen, “Efficient XML interchange (EXI) primer,” W3C, Tech. Rep., 2014.
- [8] C. Jennings and Z. Shelby, “Media types for sensor markup language (SenML) draft-jennings-senml-10,” IETF, Tech. Rep., 2013.
- [9] (2016) Arrowhead framework wiki. [Online]. Available: https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page
- [10] “Iso/iec 81346 industrial systems, installations and equipment and industrial products – structuring principles and reference designations,” ISO/IEC, Tech. Rep., 2009.

-
- [11] O. Carlsson, D. Vera, J. Delsing, B. Ahmad, and R. Harrison, “Plant descriptions for engineering tool interoperability,” in *Proceedings of IEEE INDIN 2016*, 2016.
 - [12] O. Carlsson, C. Hegedűs, J. Delsing, and P. Varga, “Organizing IoT Systems-of-Systems from standardized engineering data,” in *Proceeding IECON 2016*, Florence Italy, Oct. 2016.
 - [13] M. Albano, L. Ferreira, and J. Sousa, “Extending publish/subscribe mechanisms to SOA applications.” in *Proceedings of the 12th IEEE World Conference on Factory Communication Systems (WFCS)*, Aveiro, Portugal, May 2016.
 - [14] L. L. Ferreira, M. Albano, and J. Delsing, “QoS-as-a-Service in the local cloud,” in *Proceedings of SOCNE 2016, in conjunction with ETFA 2016*, Berlin, Germany, Sep. 2016.
 - [15] M. Albano, R. Garibay-Martínez, and L. L. Ferreira, “Architecture to support Quality of Service in Arrowhead systems,” in *Proceeding of INFORUM 2015*, Covilhã, Portugal, Sep. 2015.
 - [16] K. Nichols, S. Blake, F. Baker, and D. Black, “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,” RFC 2474 (Proposed Standard), Internet Engineering Task Force, Dec. 1998, updated by RFCs 3168, 3260. [Online]. Available: <http://www.ietf.org/rfc/rfc2474.txt>

PAPER J

Engineering of IoT automation systems

Authors:

Oscar Carlsson, Daniel Vera, Eduardo Arceredillo, Markus G. Tauber, Bilal Ahmad, Christoph Schmittner, Sandor Plosz, Thomas Ruprechter, Andreas Aldrian, and Jerker Delsing

Reformatted version of paper originally published in:

Book chapter, IoT Automation: Arrowhead Framework, Taylor & Francis, 2017.

© Taylor & Francis 2017 From IoT Automation: Arrowhead Framework by Oscar Carlsson, author Jerker Delsing, editor. Reproduced by permission of Taylor and Francis Group, LLC, a division of Informa plc.

Chapter 6 - Engineering of IoT automation systems, 2017, Daniel Vera, Eduardo Arceredillo, Markus G. Tauber, Bilal Ahmad, Christoph Schmittner, Sandor Plosz, Thomas Ruprechter, Andreas Aldrian, and Jerker Delsing, *Engineering of IoT automation systems*, Taylor & Francis, 2017.

1 Introduction

The acceptance of new technology depends to a high degree on usability for the end users. As the continued usability of technical systems depends on the engineering support throughout the life cycle, the support for engineering and continued operation is vital for successful acceptance of new technology. As the Arrowhead systems are envisioned to cater to a wide array of users in different domains of our society, there will be a need for different methodologies depending on both the requirements in different areas as well as the skills expected from the different users.

Across the domains covered by Arrowhead partners a select number of recommended methodologies have been identified and described. To further illustrate the situations where the different methodologies may be suitable, a number of scenarios have been provided. The scenarios range from deployment of numerous sensors at a minimal engineering and operation cost, through high-security, industrial applications to replacement of devices due to failure or maintenance operations.

2 Engineering of an Arrowhead compatible multi-domain facility

For very large projects involving automation, especially in projects geared towards the process industries, it is common that engineers from different disciplines are required to coordinate the designs of buildings, electrical systems, utilities, piping and instrumentation, process equipment, and automation systems. Most parts used in each individual discipline are either strictly standardised or available as commercially off the shelf (COTS), making the design more of a coordination and composition activity than free design work. The engineering process and its associated information management are described in greater detail by Yang [1].

Throughout the design and engineering phases, different tools will be used for the different disciplines. Most of these tools can be grouped together as computer-aided design (CAD) software. Many of the tools are based on or can export to databases but they are typically not made to be integrated directly. As the existing tools are very capable in their respective fields, it is proposed that they are kept separate and that engineers can keep working using the tools that are best suited for their discipline.

The interoperability of engineering tools is possible today - as long as all partners follow the same standards. However, as the Arrowhead project targets many domains and engineering tools have to cover different aspects and life cycle phases of those domains, there are many standards that could be used depending on the domains and life cycle phase. Even within a single domain there can be many standards available, as illustrated by the comparison of standards by Braaksma et al. [2].

In order to synchronise, it's suggested that the design engineers uses the Arrowhead Framework, throughout the whole process and uses the PlantDescription (Section3.1) as a reference tool for objects and systems that have been identified and how they are

related to each other. At the early stages the PlantDescription can help to coordinate their work, as the design initially is often focused on the overarching functionality and names of objects, and responsibilities of subsystems may not be fully decided yet. At this point the PlantDescription can illustrate how many subsystems may be present in each larger section and how these subsystems are identified within each discipline.

If the tools are enhanced to integrate with a PlantDescription system directly, it will allow engineers from different disciplines to populate the design with specific objects and the data can be synchronised between design tools using the PlantDescription. Throughout the design phase, all of the engineering data is still stored and maintained in the formats preferred by the engineering tools in their respective databases. The data in the PlantDescription should be limited to what objects are present in the separate databases, how they are identified, and what their relations are.

2.1 Further development of engineering tool interoperability

As many domains already follow standards relating to one or more of the aspects that this task aims to address, it seems unreasonable to make all of them follow one standard. Within the project a number of standards have been identified that all relate to engineering data to some extent. The following list includes a selection of them:

- CAEX (IEC 62424) [3]
- AutomationML [4] (IEC 62714)
- MIMOSA [5]
- OPC UA [6] (IEC 62541)
- IEC 61449 Function blocks [7]
- IEC 81346 Industrial systems structuring principles [8]
- ISO 15926 Industrial automation systems and integration [9]
- fastAPI Swedish building automation [10]
- IEC 61850 Power utility automation [11]

To alleviate the situation with several standards at the same site, there are already some initiatives on specific synchronisation between pairs of complementing standards such as collaboration between AutomationML (IEC 62714) and OPC UA (IEC 62541) [12] as one example and collaboration between ISO 15926 and Mimos [13] as another.

The Reference Architecture Model Industrie 4.0 (RAMI4.0) status report [14], a product of the German initiative Industrie 4.0 [15], is centred around the standards IEC 62890 for structuring the life cycle and value stream, combined with the two standards

IEC 62264 (ISA-95) and IEC 61512 (ISA-88) for structuring the hierarchical levels. At a more detailed level the report suggests a number of standards for different aspects. For implementation of the communication layer, the report suggests OPC-UA; for the information layer IEC 61360 (ISO 13584-42), eCl@ss, Electronic Device Description (EDD), and Field Device Tool (FDT) are suggested. Field Device Integration (FDI) is suggested as integration technology and for end-to-end engineering the report suggests ProStep iViP, eCl@ss, and AutomationML (which uses a topology based on IEC 62424).

Many of the standards concern the parametrisation of the object data, which is important for compatibility between tools within one domain, but as the target here is interoperability between tools from different domains it may be sufficient to focus on a few key parameters for each object and the different relations the objects have among themselves. Once the interoperability reaches the point of complete integration of all engineering data into a single data store, the standard ISO 18876, which establishes an architecture and methodology for integrating industrial data, could be used.

3 Component-based engineering methodology

The concepts and paradigms described here form a basis on which tools and methods for supporting component-based manufacturing system life cycle can be implemented. The procedure focuses on the engineering lifecycle of PLC based automation systems and was developed with the automation systems used for automotive engine assembly operations in mind.

In the domain of manufacturing engineering, the concept of component and component-based engineering methodologies aims at providing some level of support for re-use of pre-validated engineering data across engineering programs. The concept of component may be materialised physically by a mechatronic or IoT device integrating mechanical, electronic and software elements. However, a component is more commonly perceived as a data container that encapsulates various aspects of engineering and their corresponding datasets (e.g., mechanical engineering, control data, process data) for a sub part of a system and at a given level of granularity. The engineering methodology then consists in supporting re-configuration and re-use of components in order to accelerate the design of a complete system.

3.1 Life cycle dimensions

The life cycle of component-based automation system is defined as the composition of two life cycles that can be concurrent and/or sequential in time:

- Component engineering life cycle
- Component-based system engineering life cycle

The component and system engineering life cycles can be concurrent and/or sequential in time (depending on system and process-related constraints), but should be de-coupled

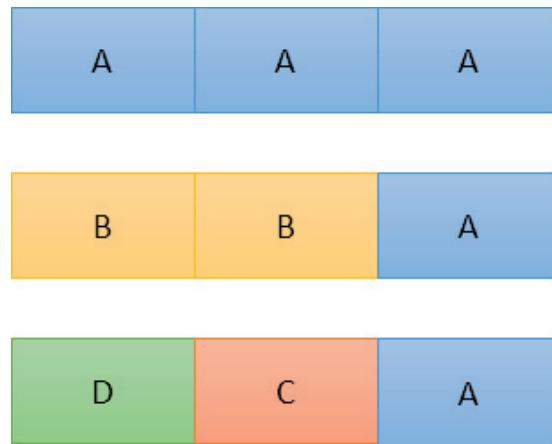


Figure 1: A component might support only one type of function or might result from the integration of several design dimensions.

as much as possible by dissociating the engineering life cycle of individual components and the composition of components into a complete system.

The component life cycle will focus on designing a set of individual components (Component Library) whose chosen levels and types of functionalities allow for systems that provide the expected characteristics to be composed. The component engineering life cycle focuses on designing, pre-testing, validating and maintaining individual Components, while the system life cycle focuses on configuration and composing components into a complete system.

The definition of the initial set of components (Component Library Design) and individual components design will evolve through time, based on the knowledge collected through multiple system engineering life cycles. An example in the domain of manufacturing system is the design of a product clamping unit (or clamping component) that might be updated based on the failures records of similar components deployed in various production lines, and/or the mechanical design may be updated to accommodate a wider range of product sizes, and/or a clamping unit and part presence sensor unit may be merged into one single component in order to facilitate the later system configuration task.

3.2 Design dimension

The intrinsic nature of systems used in the manufacturing industry (as in most technological domains) requires expertise in several domains of engineering. Typically, automation systems are electro-mechanical systems in which engineering relies on sub-design activities such as electrical, process control, hydraulic, electronic engineering, etc.

For component-based system design, the implication of having several design dimen-

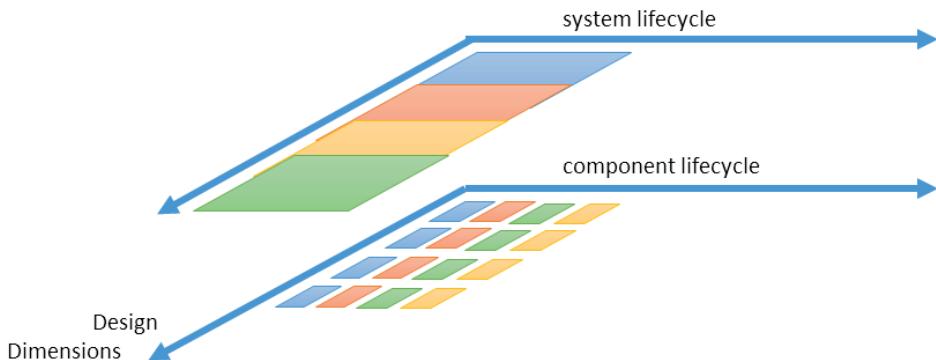


Figure 2: Design dimensions across system and component life cycles.

sions has to be considered from both component and system design perspectives. Figure 1 shows schematically how design dimensions can result in components of different natures. The nature of a component might be of a single type, i.e., support only one type of functions (e.g., type/function A), or a component might result from the integration of several design dimensions (e.g., electronic and mechanical and software etc).

The choice of building a library based on single-dimension or multi-dimension component designs, or a mix of both, might be dependent on many parameters and/or system requirements: nature of the system being designed, requirement in terms of component re-usability and system re-configurability, design commonality strategy, technological limitations, etc. However, it is critical for tools and methods to provide some form of support across all design engineering dimensions, whether at the component or system design life cycle or both. Figure 2 illustrates the possibilities with design dimensions across life cycles.

3.3 Data model

Modern system engineering activity results in and/or initiates from a set of digital data. Specific design dimensions of a system will be defined by a specific dataset expressed in one or more data formats and generated by specific engineering domains or organisations. The same design dimension might be described using different datasets and data formats (e.g., various CAD formats, different control code, etc.). The dataset and data formats used to describe a system or sub-system are defined as the data model.

For component-based system design, the data model exists at two levels of abstraction: the component and the system levels. At the component level, the data model will reflect the grouping of various design dimensions and define a dataset that includes engineering-specific data expressed in engineering-specific formats, cf. Figure 1 and 3. The component data model should also include data describing the correlation between various design dimensions (e.g., mechanical actuator position corresponding to a given logic state).

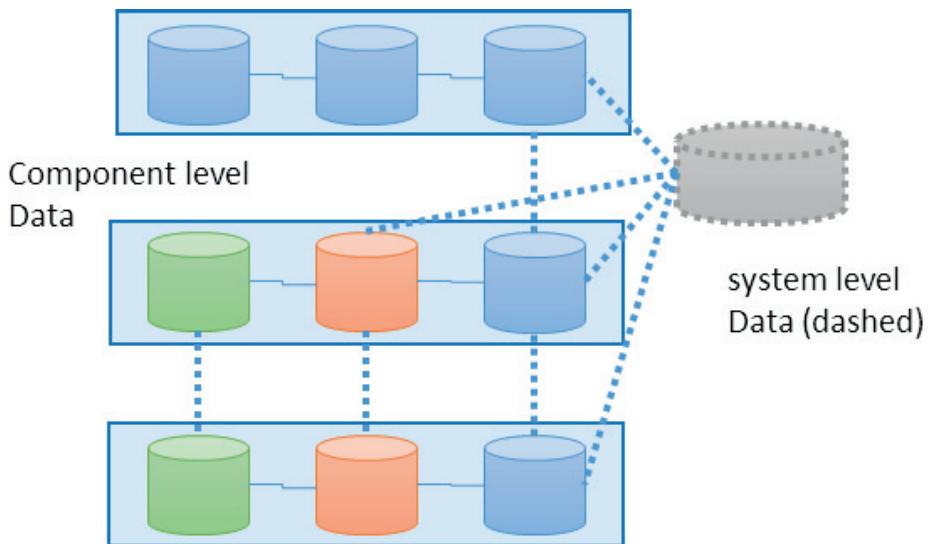


Figure 3: Component and system data for the design dimensions used, cf. Figure 1.



Figure 4: Gap between engineering and operation phases of the same system.

The system level data model will define the data required to achieve functional composition of component data (e.g., process sequence interlock) and any additional data required to support system level design dimension, their definition and functionalities.

Digital data enable computer models to be implemented for virtual design, testing, and validation of both component and component-based systems. Digital models and the virtual engineering activity that such models enable are of significant importance when designing engineering tools and methods. Typical virtual engineering tools used in manufacturing system engineering (e.g., PLM) often include a database and data management system, system engineering (i.e., engineering data editing) as well as a set of simulation environments for virtual validation of system design.

Virtual engineering is a critical enabler for achieving efficient design and validation of component systems. Data models and the virtual models used to simulate and validate components and systems at various stages of their life cycle create a new design dimension/life cycle space. One major objective when designing engineering methods and tools is to avoid the discrepancy between the virtual system definition and its physical implementation which typically occurs after system commissioning (physical implementation

and deployment), cf. Figure 4.

3.4 Design guidelines for component-based engineering tools

Following the line of design points made in the previous sections, the following general guidelines for engineering tools and methods can be stated:

- Life cycle aspects specific to component-based system design
 - It is essential to reinforce the difference between component lifecycle (component design) and system life cycle (system configuration)
 - Component and system design should not be integrated, so that system design can be achieved with minimum component re-design
 - Component and system life cycle should be coupled so that system design knowledge can be used to refine the design of component design and/or component library composition
 - Components should be readily configurable pre-tested, pre-validated sub-systems units
- Design dimensions
 - Engineering tools should provide flexibility with regard to the nature, types, and level of granularity of the components that can be designed and subsequently stored in the library
 - Engineering tools should provide a clear differentiation between component design and system design workflow and UI
 - Engineering tools should reinforce best practices (for component design and system configuration) for the type of system being considered
 - Engineering tools should provide functions adapted to specific a design dimension (i.e., engineering domains)
- Data model properties
 - Engineering tools should enable editing and storage of component and system design data in appropriate preferably open formats to enable integration with other engineering tool environments
 - Engineering tools should provide collaborative capabilities in order to enable component and system data editing by different, possibly distributed, organisations
 - Engineering tools should provide seamless integration of virtual and real component and system data editing processes

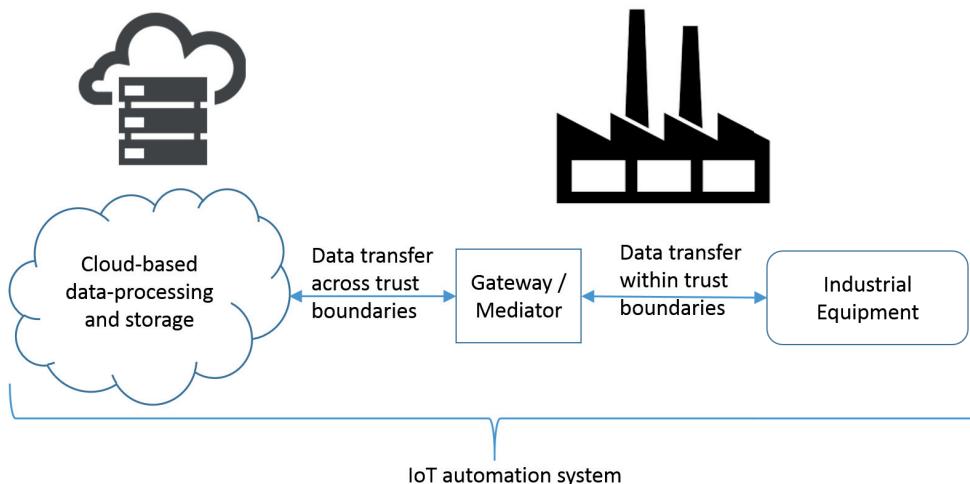


Figure 5: Basic architecture of industrial IoT automation systems.

- Engineering tools and associated databases management systems should provide the ability to maintain consistency between digital system data and the real system (after commissioning/deployment phases)
- Engineering tools should reinforce a component data model versus a system data model (access, editing, visualisation, etc).

4 Safety and security engineering of IoT automation systems

Modern IoT automation systems are being increasingly connected to the Internet, especially when cloud services are involved. This means the overall system is exposed to cyber security risks, cf. Figure 5. As automation systems are used to control industrial equipment, such cyber security threats could have an effect on the operational safety of the overall automation system. Traditionally, the system and especially the safety- and mission-critical part linked to the control of the mechanical industrial device was isolated from the Internet. Hence, while safety and reliability engineering had a long tradition in this domain, cyber security issues have played a minor role.

Therefore, when engineering an IoT automation system, both safety and security and their interplay must be considered and assured in order to guarantee a smooth operation and trust in the system.

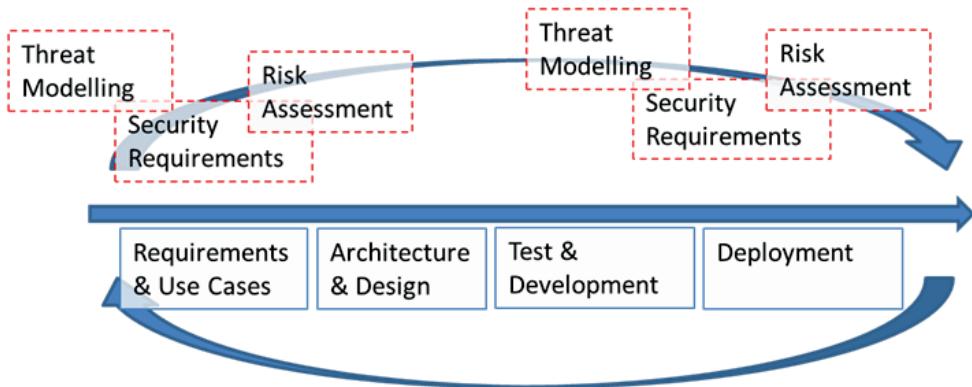


Figure 6: Security-aware system design life cycle.

4.1 General approach

Based on the methods that have been consolidated in the research phase of the project, a safety and security analysis has been made in sync with the engineering process. The methods for both safety and security analysis include

- identification of the assets of interest for the stakeholders operating the IoT automation system followed by system modelling
- identifying vulnerabilities and threats (security) and their counterparts in the safety domain (i.e., failure modes)
- ranking them using risk assessment approaches

A special focus was on threats or failures which threatened cross-domain properties, e.g., security threats which endangered the safety or reliability of operation. The result of risk assessment provides a feedback for system engineers about the security, safety, and reliability of the developed system and defines requirements for next iterations of the security and safety concept.

4.2 Performing security analysis

Parts of the methods described for security analysis on IT architectures in an M2M (machine to machine) context have been published [16]. Based on safety and security analysis methods, developed by Austria Institute of Technology, an architectural safety and security analysis on the Arrowhead pilot “smart-engine pilot” was conducted. This section details the security assessment methodology.

The individual architectural security analysis activities need to be conducted iteratively and in parallel to the development process of the entire system. Figure 6 captures some of the most important activities considered as part of our security analysis and how

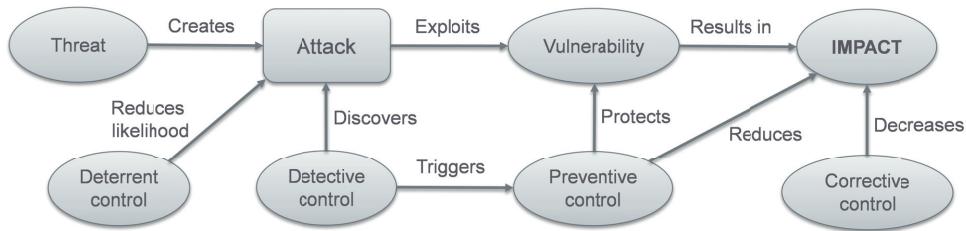


Figure 7: Terms and relations.

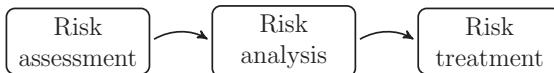


Figure 8: ISO 27005 Security Risk Management Process.

this aligns with other activities. Further security analysis related activities, e.g., the design and evaluation/testing of security features, may be added later. Some methods are defined per each of the activity groups above, but methods and standards also exist that capture multiple groups of activities. Another distinction between methods is that some are considering technical aspects in depth. Other methods address high-level technical and organisational issues.

The terms which we will be using in this document and their relations are depicted in Figure 7 and explained as follows. Threat is a possibility that a system can receive an attack. An attack which is created by a threat exploits the vulnerabilities of the system. Vulnerabilities make a system more prone to an attack by a threat or make an attack more likely to have some success or impact. The likelihood is the frequency or probability of something to occur. The impact is the result of an attack on the system which may be damage, loss of information or service, manipulation of data, etc.

Methods

The security analysis is usually performed as depicted in Figure 6, starting with threat modelling, followed by definition of requirements and ending with the risk assessment procedure. This method is along the risk assessment guideline defined in ISO 27005 standard, on which we rely when selecting the security analysis methods. The basic steps to perform risk assessment using the guideline are depicted in Figure 8 based on the ISO 27005 standard [17].

According to the ISO guideline, risk assessment consists of risk analysis and risk evaluation. Risk analysis consists of risk identification and risk estimation. The ISO standard was taken under consideration when developing the security analysis methodology used in the pilots.

Risk modelling serves the purpose of “abuse cases identification.” “Security Requirements” help to initiate security analyses but can also be a result of it. “Risk assessment”

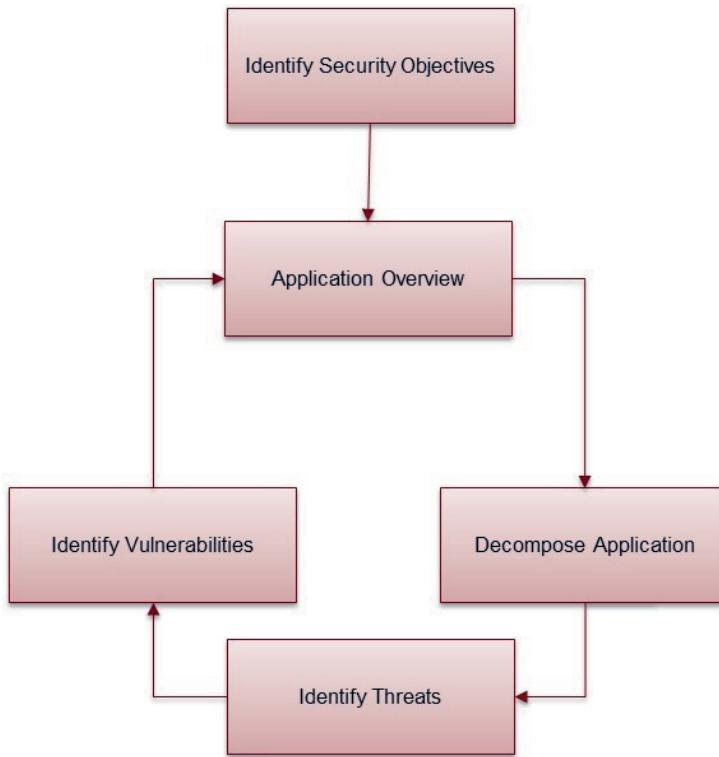


Figure 9: Microsoft threat modelling process steps.

allows linking the probability of likelihood and the effect of a threat becoming an attack to, e.g., determining which counter measure development should be prioritised. During research multiple risk modelling, analysis, and assessment methods have been investigated. The methods which have been consolidated to the security analysis methodology are detailed in the following sections.

Microsoft threat modelling process The Microsoft threat modelling process originated from secure code writing efforts at Microsoft in the early 2000s. In 2006 the Microsoft security development life cycle was developed [18]. The threat modelling process consists of the steps shown in Figure 9.

In the initial step the security objectives need to be defined. Decisions made in this step have a great influence on the following steps in the analysis process. The goal of the next two steps, application overview and decompose application, is to gain a good understanding of the underlying system architecture. As an output of these two steps, architecture diagrams on different levels of detail are created. After having a clear understanding of the system architecture, potential threats to system security and

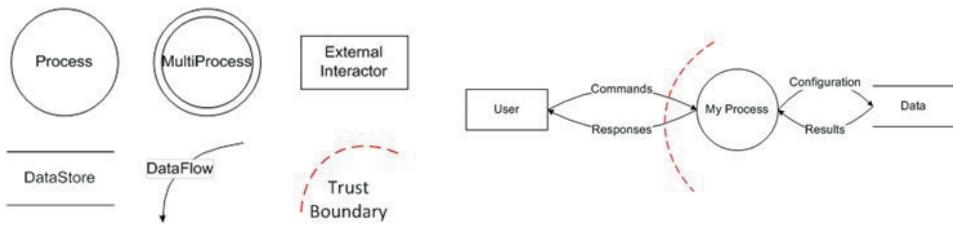


Figure 10: Modelling objects used in a data flow diagram.

already known vulnerabilities are collected.

The decomposition of the application is normally achieved by developing a data flow diagram (DFD), which consists of processes, data stores, boundaries, and the data flows between them. For identifying threats and vulnerabilities the so-called STRIDE [19] method is applied on each DFD component; each letter stands for a category of threats: Spoofing, Tampering Repudiation Information disclosure, D.o.S, and Elevation of privilege.

Results In order to use Microsoft Threat Modelling with STRIDE, one needs to know which assets are concerned. This can be discovered by interviewing system experts. Threats against assets are identified when using STRIDE after decomposing a system in data source, processes, data flow, and interactors in a DFD. The most important DFD components are shown in Figure 10 and in a simplistic use case example.

The output or result is a comprehensive list of categorised threats. The identified threats and vulnerabilities can then be ranked with any risk assessment approach to derive the most important threat to deal with.

Usage in Arrowhead The MS threat modelling process is currently applied to a work package pilot. The methodology presented allows application to other pilots as well. Four cases of high-level architectures have been developed as a starting point for evaluating the security analysis. This means that by categorising the use cases in such a way, vulnerabilities/threat examples can be used between pilots. The four cases are

1. A measurement device talking directly to a back-end system which stores data in a “cloud” or central back-end (called “cloud” for simplicity hereafter)
2. A measurement device talking via a proxy (for multiple devices) at the client side to a back-end system which stores data in a “cloud”
3. A measurement device talking directly to a back-end system which stores data in an external “cloud”
4. A measurement device talking via a proxy (for multiple devices) at the client side to a back-end system which stores data in an external “cloud”

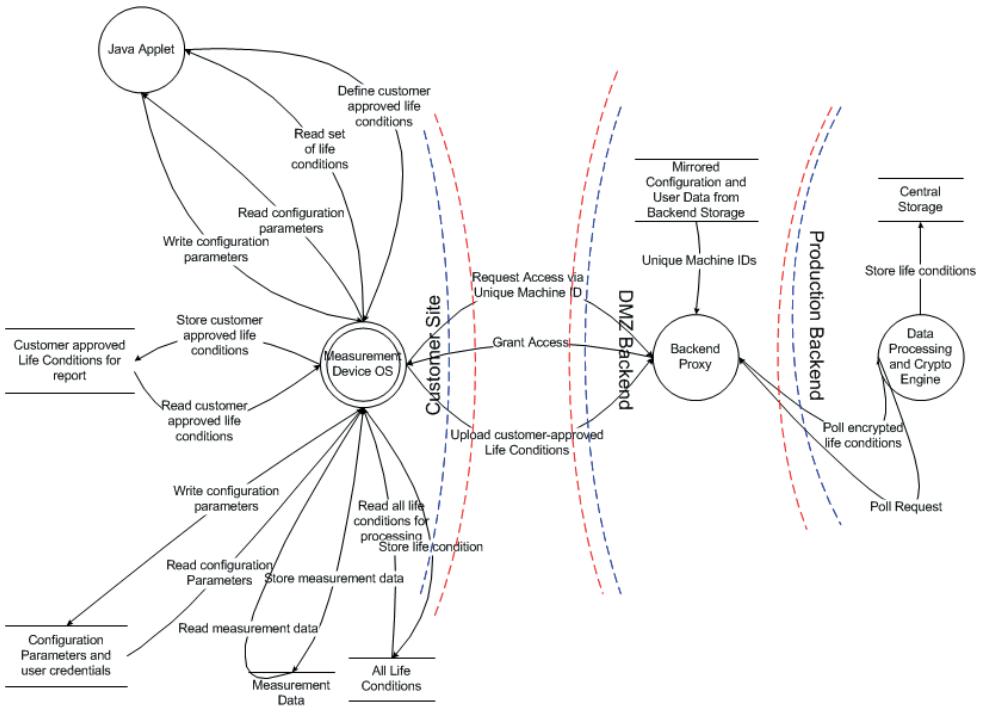


Figure 11: Data flow diagram for end-to-end security use case.

Application overview and decomposing In the course of the decomposition process, a data flow diagram (DFD) has been developed together with system experts of the use case illustrated in a DFD in Figure 11 (a measurement device talks via a proxy with a cloud back end).

Identify threats and vulnerabilities STRIDE is an established method which originates from the Microsoft secure software development life cycle guide lines. It is a method for threat identification and categorisation.

The category of threats can be identified by looking at the system components and the flow of information between them in a data flow diagram as part of the STRIDE method.

In STRIDE the elements of the data flow diagram are affected by different kind of threats defined by an element type-threat matrix. Based on this and the data flow diagram, the tool generates the list of threats. A group of engineers has to discuss these threats and decide which is relevant in their use case, and decide mitigation methods.

Advantages and conclusion The Microsoft threat modelling method is easy and straightforward to use especially with the available supporting tools (e.g., MS SDL). Potential improvement of this approach includes the considerations of further vulnerability catalogues, e.g., National Vulnerability Database (NIST), IT-Grundschutzor attack patterns (<http://capec.mitre.org/>).

DREAD methodology DREAD is a simplistic methodology for linking threat (categories) to weight motivated by the expected impact. DREAD stands for the individual issues to consider for each individual threat. DREAD stands for

- **D**amage potential (how severe the damage is)
- **R**eproducibility (how easily the attack can be reproduced)
- **E**xloitability (how hard it is to work out how to attack)
- **A**ffected user
- **D**iscoverability (how easily the vulnerability can be detected)

Each individual threat is awarded a score for each DREAD issue and the average determines the risk value (high values mean high risks). It is often used in combination with STRIDE and part of the Microsoft security development life cycle [18].

Results To achieve meaningful results it is recommended that each individual vulnerability or threat identified with, e.g., STRIDE is rated for each category of DREAD. Ratings may come from a pre-defined range (0, 5, 10 are the most common ones). Pre-definition in this context also means that the security and system experts have to agree on a consistent rating/scoring scheme (e.g., what low Damage Potential means) prior to application. Scores per category are given under the assumption that the threat/attack has been launched successfully for each vulnerability. The overall risk rate calculation formula is the following:

$$\text{Rating} = (D + R + E + A + D) / 5$$

Usage in Arrowhead The MS threat modelling process with STRIDE can be applied in combination with DREAD to pilots. This was conducted as part of the design phase. Individual threats have been ranked with DREAD to support finding the security focus in a first pilot iteration.

Conclusion This simplistic approach was easy to implement using Excel spreadsheets. It is recommended to conduct the approach in pairs to avoid biased opinions.

Combined ETSI and STRIDE methodology

We have combined STRIDE threat identification with risk assessment methods standardised by the European Telecommunications Standards Institute (ETSI) [20]. This security analysis methodology consists of several steps, which are depicted in Figure 12. The analysis is a security risk assessment process. Which provides impact- and likelihood assessment of sub-systems which is aggregated to a final global security analysis results.

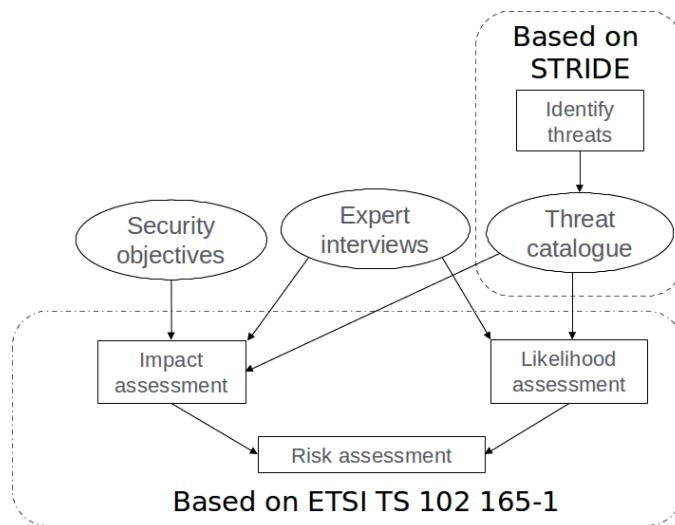


Figure 12: Security analysis methodology.

The first step defines security objectives which are to be considered during the whole assessment. The next step is to build a model of the system and identify possible threats. Hence, Microsoft's STRIDE methodology is used for this step to create a catalogue of the most relevant threats which are then further evaluated to assess their impact and likelihood of the risk involved. In order to support our analysis, as inputs for assessing encountered threats, interviews with system experts have to be conducted through questionnaire, survey, or personal consultation.

Defining security objectives According to the ISO 27005 standard, security analysis has to be performed in the planning stage of the system. At first the security objectives which are relevant for the system operation have to be determined. Types of security objectives are

- Confidentiality: Data is only available to the people intended to access it.

Table 1: Affected security objectives by threat types

Threat type	Affected security objective
Spoofing	Authentication
Tampering	Integrity
Repudiation	Nonrepudiation
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorisation

- Integrity: Data and system resources are only changed in appropriate ways by appropriate people.
- Availability: Systems are ready when needed and perform acceptably.
- Authentication: The identity of users is established (or accepting anonymous users allowed).
- Authorisation: Users are explicitly allowed or denied access to resources.
- Nonrepudiation: Users cannot perform an action and later deny performing it.

Generating the threat catalogue Security risk analysis is performed based on ISO guidelines. It is composed of two steps: risk identification and risk assessment. For risk identification the Microsoft STRIDE method has been used. Each of the threat categories of STRIDE violates one of the above mentioned security objective, as indicated in Table 1.

Impact assessment In order to estimate risk of the identified threats, their impact and likelihood need to be assessed. Impact assessment evaluates the consequences of attacks performed by the threats which directly influence the risk involved. The level of impact can be measured multiple ways, such as

- Magnitude of loss
- Cost of losing data
- Level of damage
- Cost of unavailable service
- Cost of repair
- Time needed for repair
- Cost of gathering the data again
- Indirect costs (trust, reputation)

Table 2: Scale, detectability and recoverability assessment

		Detectability and Recoverability		
		Easy	Medium	Hard
Scale levels	Node	Minor	Minor	Moderate
	Local	Moderate	Significant	Significant
	Area			
	Network			
Enterprise	Network	Moderate	Significant	Significant

Scale, detectability, and recoverability assessment In the impact assessment method, impact depends on two factors, the scale level, and detectability and recoverability levels. The scale level expresses whether the attack only targets and has impact on a particular node or element, and affects the operation of a local part of a system or network, or maybe even the whole system or network is affected by an attack. Therefore three values can be assigned to this property (see also Table 2):

- Node: Only the targeted node is affected by the threat; a fault is not propagated to the adjacent nodes/elements
- Local (Area) Network: the threat affects the operation of the local network
- Enterprise Network: the whole network is affected by the threat

The detectability and recoverability property expresses that in case of an attack how hard it is to first detect the attack, avert it, and then recover the system to a state that was present before the attack. These possible values can be assigned to it:

- Easy: Detecting the attack is easy, recovery is quick
- Medium: detecting the attack is moderately difficult or the recovery takes time, possibly requires user intervention, and the exact state cannot be recovered
- Hard: detecting the attack is difficult, distinct algorithms have to be implemented, recovery is not straightforward, requires user intervention, and there is possible data loss involved.

Asset impact and attack intensity assessment In the ETSI standard the overall impact level is influenced by two factors: the impact on the asset and the attack intensity. An asset is basically an equipment or resource which has value to the organization or the company. The impact of a threat to an asset can be on the following three levels:

- Low: The concerned party is not harmed very strongly; the possible damage is low (value 1)

Table 3: Motivation and difficulty assessment.

Motivation	Difficulty	None	Solvable	Strong
Low	Unlikely	Unlikely	Unlikely	
Moderate	Likely	Possible	Unlikely	
High	Likely	Likely	Unlikely	

- Medium: The threat addresses the interests of providers/subscribers and cannot be neglected (value 2)
- High: A basis of business is threatened and severe damage might occur in this context (value 3)

There are also three levels defined for the attack intensity as follows:

- Low: Single instance of attack (value 0)
- Medium: Moderate level of multiple instances (value 1)
- High: Heavy level of multiple instances (value 2)

The two factors considered here are the number of attack instances and the time interval between the attacks. Based on these the impact is calculated by adding the impact and attack intensity values together.

Likelihood assessment The second part needed for the risk assessment is estimating the likelihood of a threat to be exploited and an that an attack is made to the system.

Two approaches have been considered for likelihood assessment, one based on motivation and difficulty assessment described in [21], the other based on time, expertise, opportunity, and equipment assessment described in ETSI TS 102 165-1 standard [20].

Motivation and difficulty assessment Difficulty of performing an attack can be (see also Table 3)

- Strong: Security mechanisms that currently may not be defeated because some theoretical elements needed for perpetrating an attack upon them are missing
- Solvable: Security mechanisms that may be countered or have been defeated in a related technology
- None: A precedent for the attack already exists

Table 4: Attack likelihood factor scoring.

Factor	Range	Value
Time (1 point per week)	≤ 1 day	0
	≤ 1 week	1
	≤ 1 month	3
	≤ 3 months	13
	≤ 6 month	26
	> 6 month	attack potential is beyond high
Expertise	Layman	0
	Proficient	2
	Expert	5
Knowledge	Public	0
	Restricted	1
	Sensitive	4
	Critical	10
Unnecessary/unlimited access	Unnecessary/unlimited access	
	Easy	1
	Moderate	4
	Difficult	12
	None	attack path is not exploitable
Specialised	Standard	0
	Specialised	3
	Bespoke	7

Table 5: Vulnerability rating.

Attack potential values	Resistant to attacker with attack potential of:
0–2	No rating
3–6	Basic
7–14	Moderate
15–26	High
≥ 26	Beyond high

Table 6: Risk assignment table.

Value	Risk	Explanation
1, 2	Minor	No essential assets are concerned, or the attack is unlikely. Threats causing minor risks have no primary need for countermeasures.
3, 4	Major	Threats on relevant assets are likely to occur although their impact is unlikely to be fatal. Major risks should be handled seriously and should be minimized by the appropriate use of countermeasures.
6, 9	Critical	The primary interests of the providers and/or subscribers are threatened and the effort required from a potential attacker to implement the threat(s) is not high. Critical risks should be minimised with highest priority.

ETSI likelihood assessment According to the standard, this is influenced by five factors. Based on these factors a scoring scheme is defined according to Table 4.

The scores of these factors are assessed and summarised giving the overall score of the attack likelihood, which is mapped to vulnerability rating according to Table 5.

Risk Assessment The risk score is calculated by multiplying the scores given to the likelihood and impact levels defined in the previous chapter. Based on this score, the risk is determined according to Table 6.

The next step is to assign the risks of the threats of the affected security objective types. In the ETSI standards the following security objective types have been defined:

- Interception
 - Eavesdropping: A breach of confidentiality by unauthorised monitoring of communication
- Manipulation
 - Masquerade (spoofing): The pretence of an entity to be a different entity. This may be a basis for other threats like unauthorised access or forgery
 - Loss or corruption of information: The integrity of data (transferred) is compromised by unauthorized deletion, insertion, modification, reordering, replay, or delay
 - Unauthorised access: An entity accesses data in violation of the security policy in force.
 - Forgery: An entity fabricates information and claims that such information was received from another entity or sent to another entity.

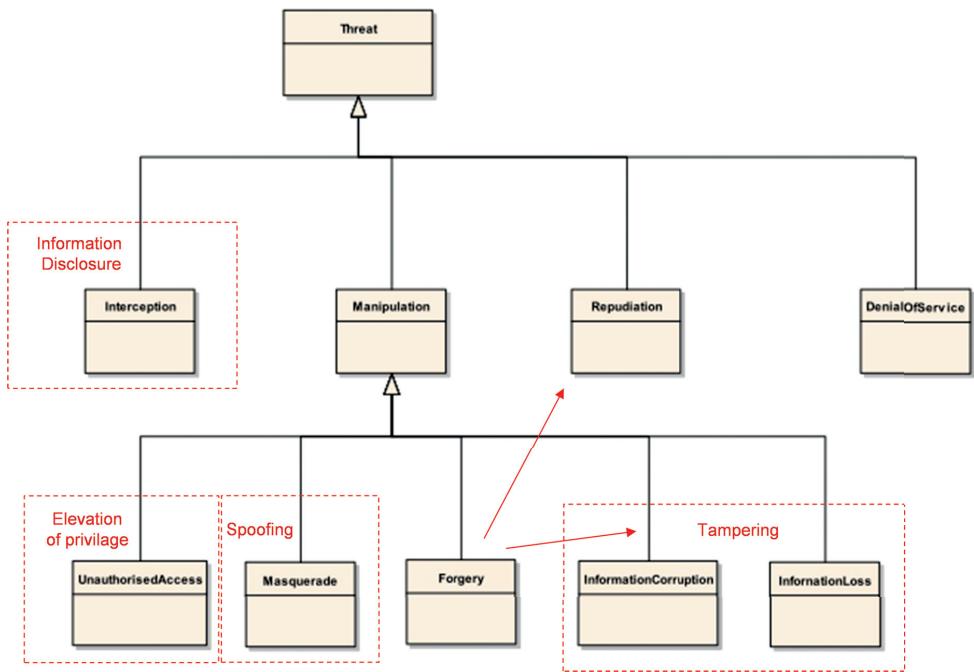


Figure 13: Assigning STRIDE threat categories to ETSI categories.

- Repudiation: An entity involved in a communication exchange subsequently denies the fact.
- Denial of service: An entity fails to perform its function or prevents other entities from performing their functions.

These threat types differ a bit from the STRIDE threat types, but the assignment between them is quite straightforward (see Figure 13).

The reasons for having the threat categories for the individual threats is that ETSI defines which of the security objectives are affected by an attack. To wrap up, the threats generated by the STRIDE method based on the DFD are categorized into one of the categories in the STRIDE acronym. These categories can be translated into ETSI threat types.

4.3 Safety analysis

FMEA/FMECA (Failure Mode Effect Analysis/Failure Mode Effect, and Criticality Analysis)

FMEA - Failure mode effect analysis FMEA is used to identify potential failure modes, determine their effect on the operations of the product, and identify actions to mitigate failures. FMEA is a bottom-up, inductive analytical method which may be performed at either the functional or component/sub-system level. A successful FMEA activity helps the developer team to identify potential failure modes based on past experience with similar products or processes, and enables them to design those failures out of the system with the minimum of effort and resource expenditure, thereby reducing development time and costs. It is widely used in manufacturing industries in various phases of the product life cycle and is now increasingly finding use in the service industry. FMEA can be applied to hardware as well as software, and it allows a quantitative approach based on known component reliabilities as well as a qualitative one, where no reliability figures are available but experts judge based on their experience.

Figure 14 shows the basic steps for performing an FMEA analysis. Based on a system description, the functions of a system are identified. One function is selected and the failure modes of the function are identified. For a failure mode, the effects and causes and the probability are identified. This is repeated for all potential failure modes and all functions. Based on the basic FMEA approach, different extensions were developed.

FMECA - Failure modes, effects, and criticality analysis FMECA extends FMEA by including a criticality analysis, which is used to chart the probability of failure modes against the severity of their consequences. The result highlights failure modes with relatively high probability and severity of consequences, allowing remedial effort to be directed where it will produce the greatest value. FMECA tends to be preferred over FMEA in space and North Atlantic Treaty Organization (NATO) military applications, while various forms of FMEA predominate in other industries.

FMVEA - Failure Modes, Vulnerabilities, and Effects Analysis FMEA approaches have a long history; they were first applied to military use cases and developed for hardware risk analysis. They have since successfully been applied to software, processes, and even security [22]. Failure effects describe the unordinary or unwanted states of the system, e.g., the impact, while failure modes, which produce the impact, are similar to an attack, but can also be an unexpected event or behaviour. Failure modes are categorized into criticality and probability levels. The results are assessed in a table and the risks evaluated, see such example in Table 7.

Application in Arrowhead FMEA analysis was applied on the end-to-end security use case. In order to ease the process and combine safety and security assessment, FMEA was applied to the dataflow model, generated for the security analysis. While

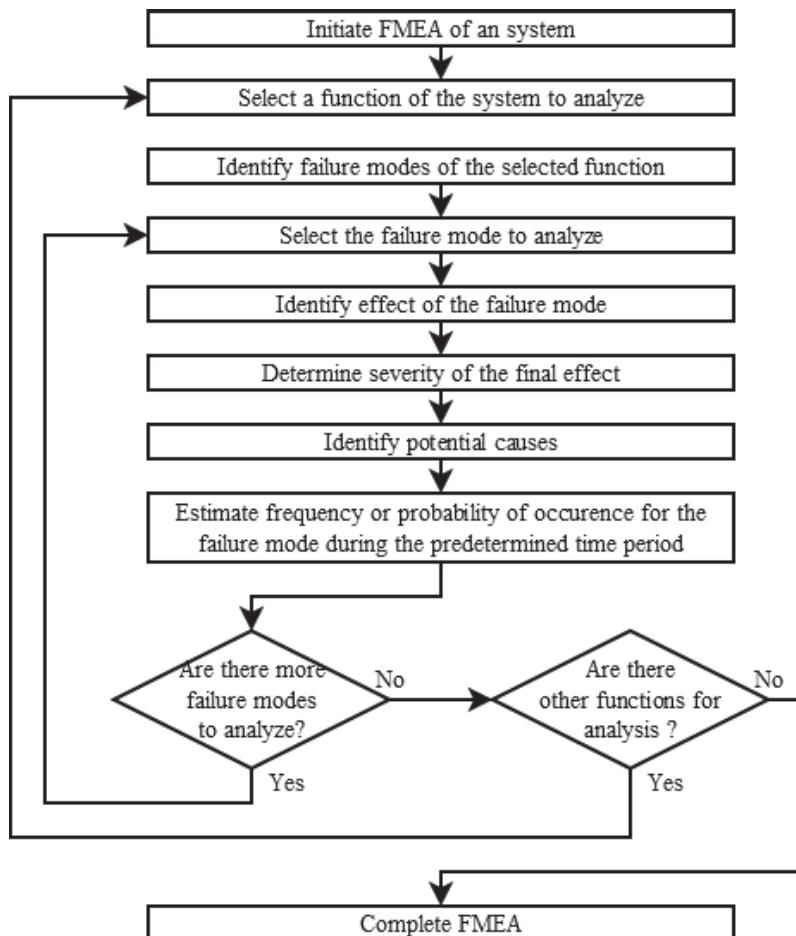


Figure 14: FMEA methodology.

FMEA was originally developed for hardware, it also generated useful results when applied to the dataflow model. We developed, based on a literature study, a list of generic failure modes suitable for elements in a dataflow diagram. We researched different lists of generic failure modes for industrial systems and developed a list which divides the failure modes into failure modes for input/output functions and processing units. This enables the application of FMEA to a dataflow diagram of a system.

Failure Modes for Input/Output Functions:

- Missing Data: Lost message, data loss
- Incorrect Data: Inaccurate data, spurious data

- Timing of Data: Obsolete data, data arrives too soon/late for processing
- Extra Data: Data redundancy, data overflow

Failure Modes for Processing Units:

- Stops with message: The processing units stops
- Obviously wrong results: The program runs, producing obviously wrong results
- Wrong results: The program runs, producing apparently correct but in fact wrong results

Failure Modes for Data Storage:

- Missing Data: data is missing
- Incorrect Data: Inaccurate data, spurious data
- Extra Data: Data Overflow

In order to analyse the effects of a failed function, it is necessary to determine, besides potential failure modes, in which system state the failure happens. System states in this

Table 7: Consequences to persons or environment.

Class	Severity level	Consequences to persons or environment
IV	Catastrophic	A failure mode which could potentially result in the failure of the system's primary functions and therefore cause serious damage to the system and its environment and/or personal injury.
III	Critical	A failure mode which could potentially result in the failure of the system's primary functions and therefore cause considerable damage to the system and its environment, but which does not constitute a serious threat to life or injury.
II	Marginal	A failure mode, which could potentially degrade system performance function(s) without appreciable damage to the system or threat to life or injury.
I	Insignificant	A failure mode which could potentially degrade the system's functions but will cause no damage to the system and does not constitute a threat to life or injury.

case relate to possible states for the system under analysis. In order to define the severity of identified effects, a severity classification based on IEC 60812 was utilised.

Based on the results, a number of high-level system goals were defined which should be considered and implemented by the system in order to ensure dependable operation.

5 Engineering scenarios

5.1 Scenario: Efficient deployment of a large number of IoT sensors

The use of sensor devices to collect information from the environment is a common technique for improvements in many areas of society nowadays. However, even if the potential benefits of using IoT sensor networks for sensing and monitoring the environment have been widely claimed, their deployment still requires a group of skilled staff. The networks also often require, long time for system setup and require regular maintenance.

The Plug & Play & Forget paradigm aims to provide techniques, methods, and tools to fulfil the characteristics of low-cost and easy deployment, operation, and maintenance of IoT sensor networks. The PPF philosophy contributes to the economical feasibility of a WSN solution by reducing the system setup time, the need of an expert team to deploy the elements, and the requirement of regular maintenance.

IoT sensor network deployment tool

The development of a deployment tool for IoT sensor networks stems from a need for efficient and understandable network deployment. This kind of tool should accept heterogeneous inputs such as the surrounding environment (in terms of the morphology of the site), maximum distances or possible physical obstacles, to suggest optimum locations for efficient deployment of network elements with respect to QoS, low interference, low power consumption, better signal propagation, accessibility, etc.

A simulation tool has been developed to determine the optimum location of sensor and relay nodes from application specific requirements. For a given plant area topology, the simulation tool gives advice in the placement of the sensor nodes and automatically calculates the location of the necessary relay nodes.

Figure 15 shows the appearance of the main tool, whose functionalities will be explained next.

As an application example, let us consider a rectangular plant with four columns inside, as represented in Figure 16. For that area, a grid pattern is considered indicating the candidate node locations, cf. Figure 17. The less distance we consider between grid points, the more accurate our simulation results will be, at the expense of higher calculation cost.

The simulation tool calculates the network deployment in the following three steps:

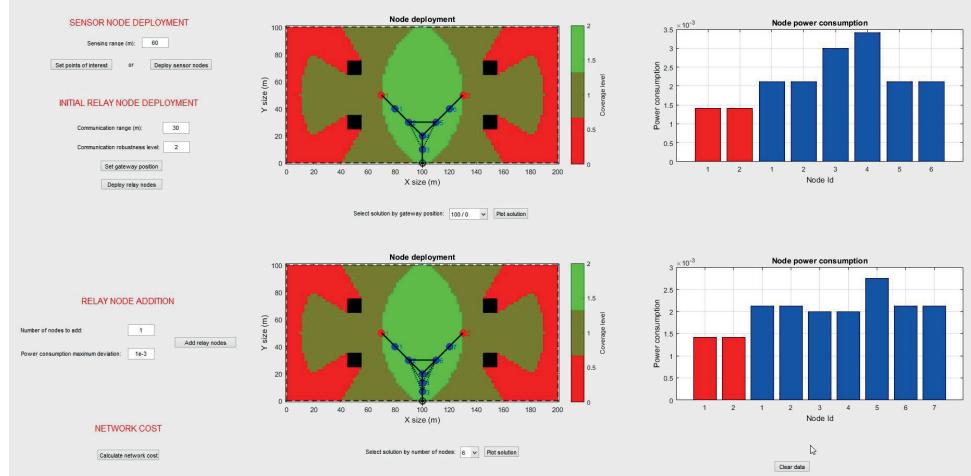


Figure 15: IoT sensor network deployment tool.

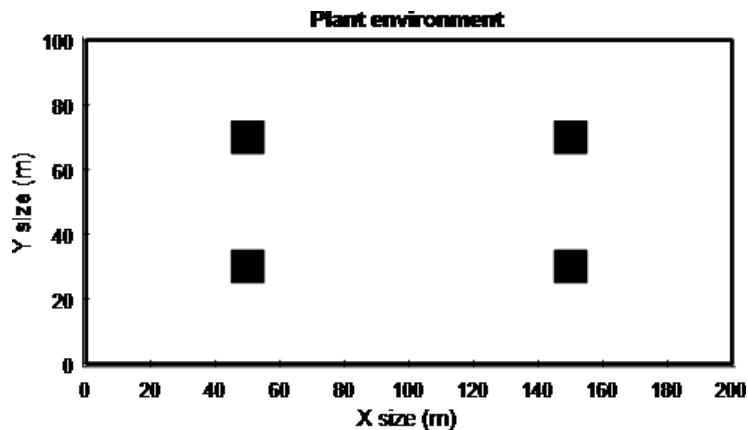


Figure 16: Plant environment.

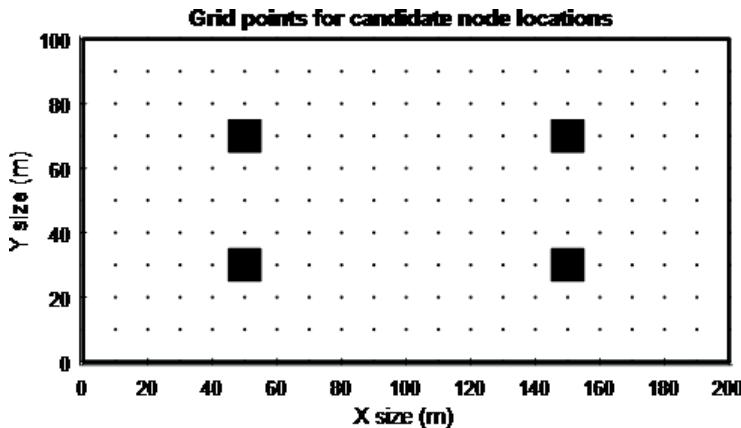


Figure 17: Available locations for node positioning.

Sensor node deployment Sensor node locations can be entered directly, or alternatively, points of interest can be entered to get some advice on the sensor node placement. If the points of interest are introduced, a sensing coverage map is represented from the points of interest with the aim of giving advice in the consequent sensor node deployment. For this calculation, nodes' sensing range and obstacle locations are considered. A point in the plant will be covered by a sensor node if, and only if, the two points lie within the sensing range and there is no obstacle between them.

Figure 18 shows the target point (or points of interest) input tool. The resulting sensing coverage map is represented in figure 19. This coverage map gives a clear idea of the locations for the minimum number of sensor nodes needed to cover all the target points. A location with a sensing coverage redundancy of 4, for instance, means that from that location four target points are covered at the same time. In this example, it is evident that only two sensor nodes would be needed to cover all the points of interest.

Figure 20 shows the sensor node input tool, and Figure 21 the resulting sensing coverage map. This is the actual sensing coverage map, which is calculated from the sensor node points. It can be seen how all the target points are covered by at least one sensor node.

Alternatively, the sensor node locations can be entered directly. In the sensor node input tool of Figure 22, no coverage advice is given. The resulting sensing coverage represented in Figure 23 is the same as that obtained in figure 21.

Initial relay node deployment Once the sensor node locations are determined, relay nodes are added in order to ensure robust connectivity. Every sensor node in the network has to be connected with the gateway directly or by means of relay nodes. To determine whether two nodes are connected or not, an effective distance is calculated (the distance through an obstacle is increased by a given scaling factor). It is considered that two points are connected with each other if the effective distance between them lies inside

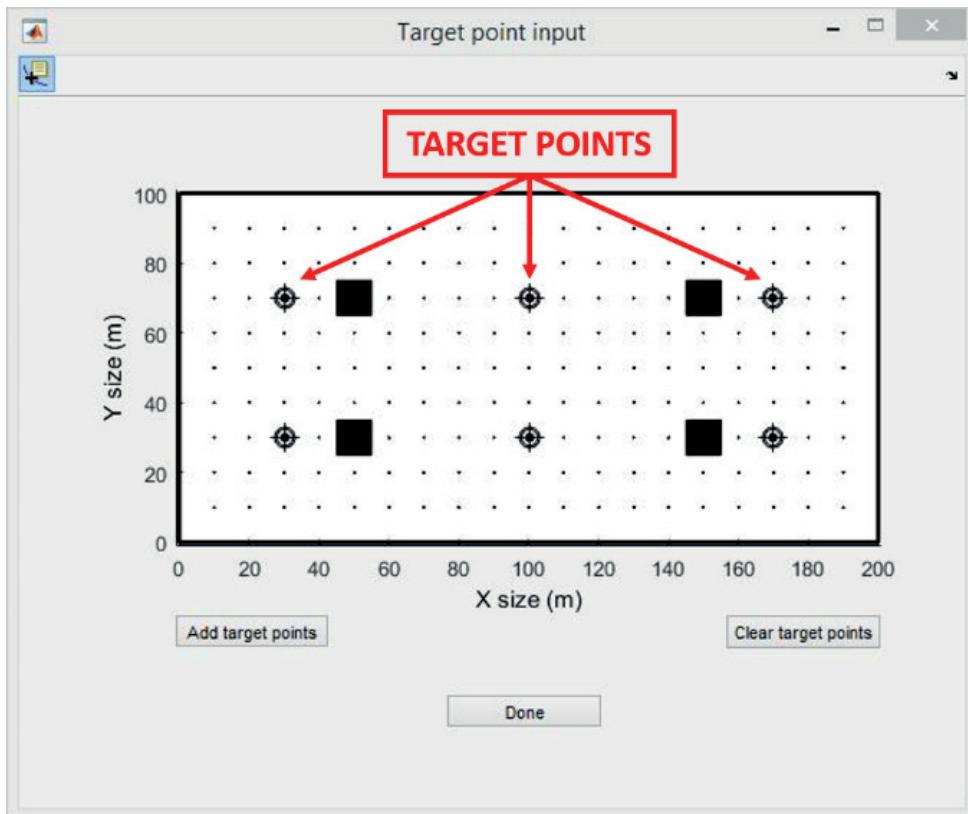


Figure 18: Target point input tool.

the communication range.

After determining the gateway position (Figure 24), the relay node locations are automatically calculated. The developed algorithm minimises the number of relay nodes by means of sharing relay nodes between different sensor nodes as far as possible. Figure 25 shows the resulting scheme for a given example. Sensor node points are represented as red points and relay node points as blue points.

Relay node addition After the initial node deployment, more relay node points can be added in order to improve network lifetime. This is done by adding more relay nodes near those with higher relaying workload, so that the workload and the lifetime of each node are more evenly distributed.

In Figure 26 the initial node deployment (with the minimum number of nodes) is represented. The bar graph of Figure 27 represents the resulting nodes' power consumptions red bars correspond to sensor node power consumption, while blue bars correspond

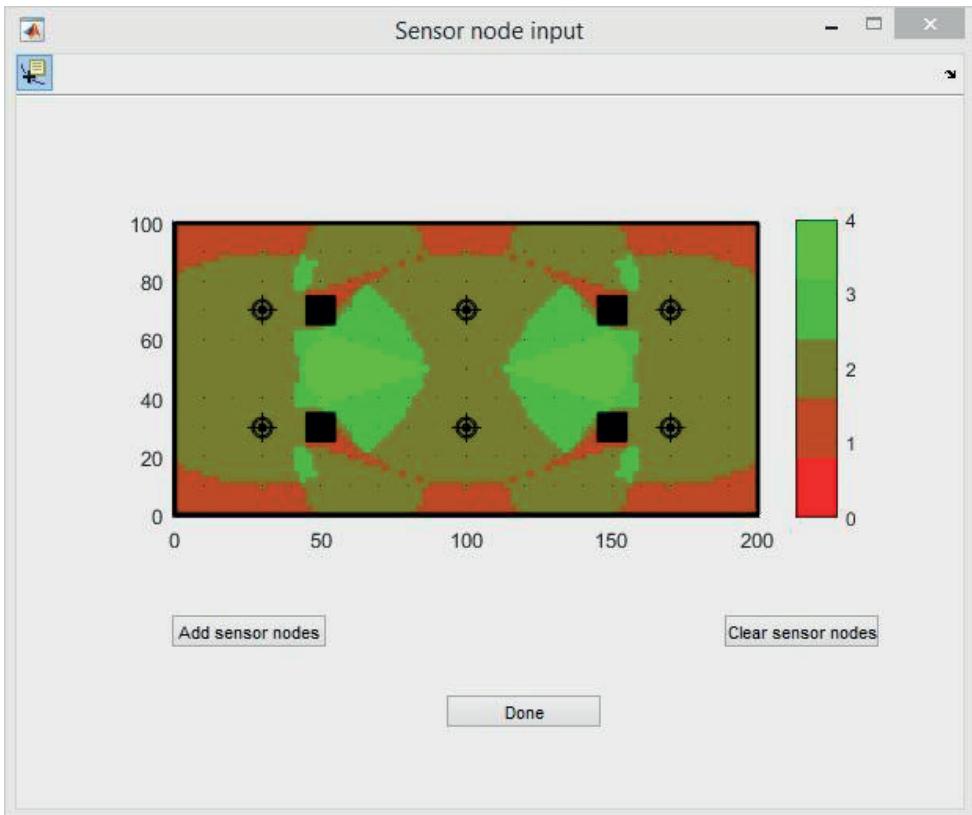


Figure 19: Sensor node input adviser tool.

to relay node power consumption. Node power consumption are calculated from nodes' transmitting and receiving distances. In Figure 28 a redeployment with an extra relay node is represented. From the power consumption represented in Figure 29, it can be seen how adding an extra relay node contributes to homogenising the nodes' power consumption.

Cost of wireless sensor network

One of the main objectives of the deployment tool is to calculate different deployment solutions in order to determine which solution is the more cost effective. Obviously, a solution with the minimum number of nodes will be the one with less material cost, but, at some time, it may become more expensive in terms of maintenance cost, as it would be necessary to replace some nodes' batteries more often.

A WSN cost tool has been developed to calculate the overall network deployment

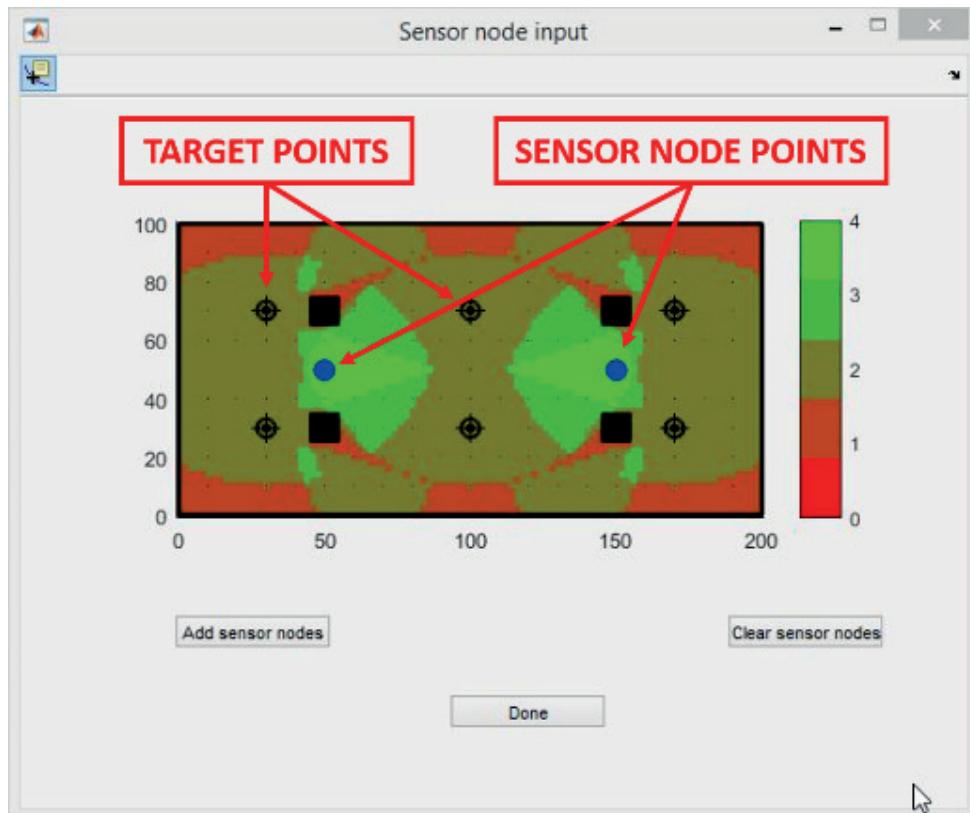


Figure 20: Sensor node input tool.

cost. The contributions considered are material cost, deployment cost, and travel cost. For each solution (that is, solutions with different numbers of nodes) two separate costs have been calculated. In the first case, each node is replaced or maintained at the time its battery runs out, while in the second case, all nodes are replaced at the time the first maintenance service is required.

Figure 30 shows the developed cost calculation tool, where specific material and deployment costs can be assigned for sensor and relay nodes (as well as for the gateway). Nodes' power consumption and battery capacity are also required in order to estimate a lifetime derived cost (nodes have to be redeployed as their batteries run out).

For each solution in Figure 26 and Figure 28 (with six and seven relay nodes), two separate costs are calculated: one for “single node maintenance” and the other for “whole network maintenance.” The represented cost is the cumulative sum of the different contributions as indicated in the legend. For the cost parameters considered, it can be seen how the seven relay node solution becomes cheaper in time as nodes in the six node

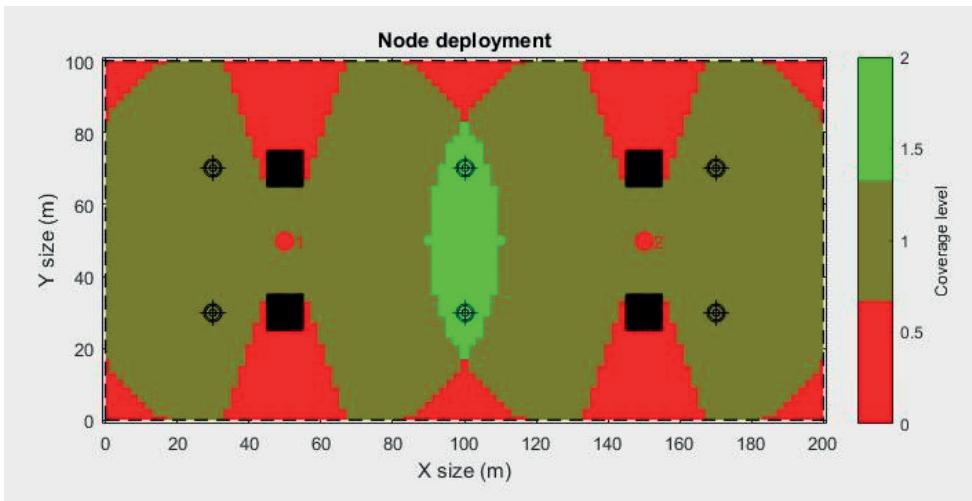


Figure 21: Sensing coverage map.

solution have to be replaced more often. Furthermore, it can be seen that “whole network maintenance” becomes cheaper than “single node maintenance” as travel cost has more impact than redeployment cost.

5.2 Scenario: Swift deployment and configuration

This scenario reflects the situation where a large number of devices are to be installed in a network where they are to perform different tasks depending or have different configurations depending on the physical location for each device.

The devices are assumed to have identical hardware and identical software preloaded from a factory, workshop, or back office, the only differences between devices are their network Media Access Control address (MAC address) and their Serial Number (S/N). The preloaded software contains the required security measures to allow the device to connect to the Arrowhead Framework core systems and to use a minimal set of services.

The information describing the different tasks and configurations is stored in a network connected storage area accessible from all locations where devices are to be installed and in a format that the devices are able to interpret. The generation and management of this information is not covered in this scenario.

Step-by-step general deployment procedure

Initially (after engineering but before deployment) there are the following systems and devices:

- A generic, configurable device, ready to be deployed

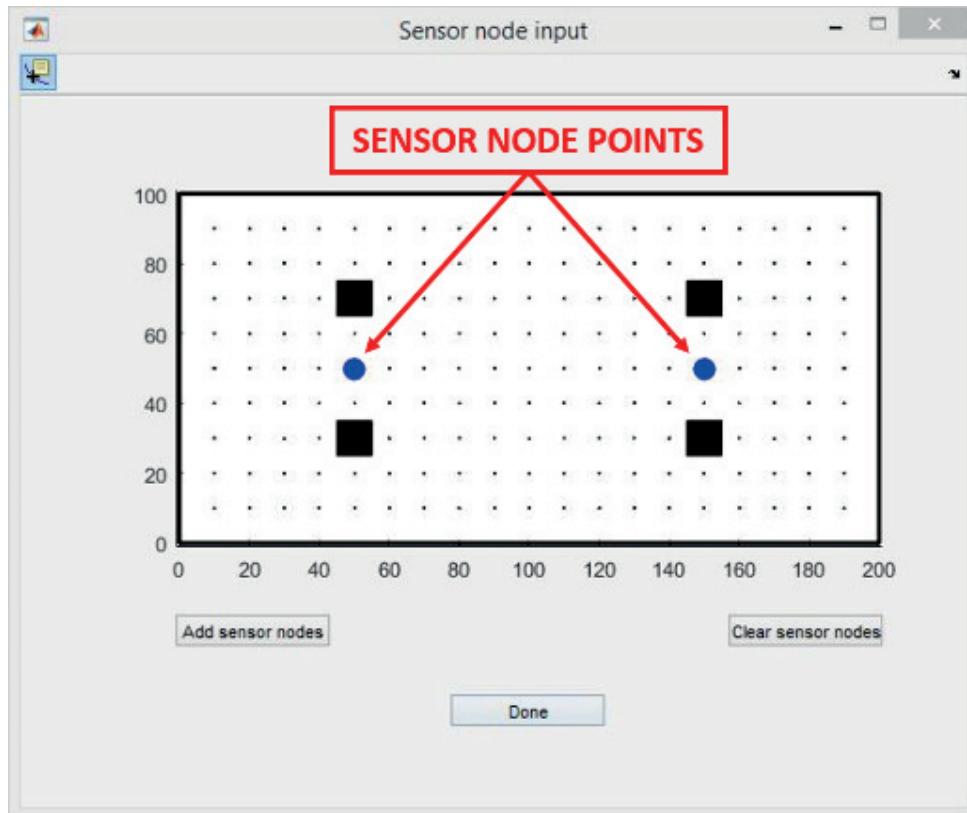


Figure 22: Sensor node input tool.

- A node in the PlantDescription representing the device that is to be introduced.
- A node in the PlantDescription representing each system on the device is intended to fulfil, each of the system nodes has at least one link to associate it with the device node.
- A configuration in the Configuration store associated with the device node in the PlantDescription. This configuration contains enough information to allow the device to host the desired systems and the NodeId by which each system node is identified in the PlantDescription.
- An identifier that is to be transferred to the device during deployment/bootstrap-ping that allows the Configuration system to associate the device with the device-node identifier from the PlantDescription. This may be achieved by using the

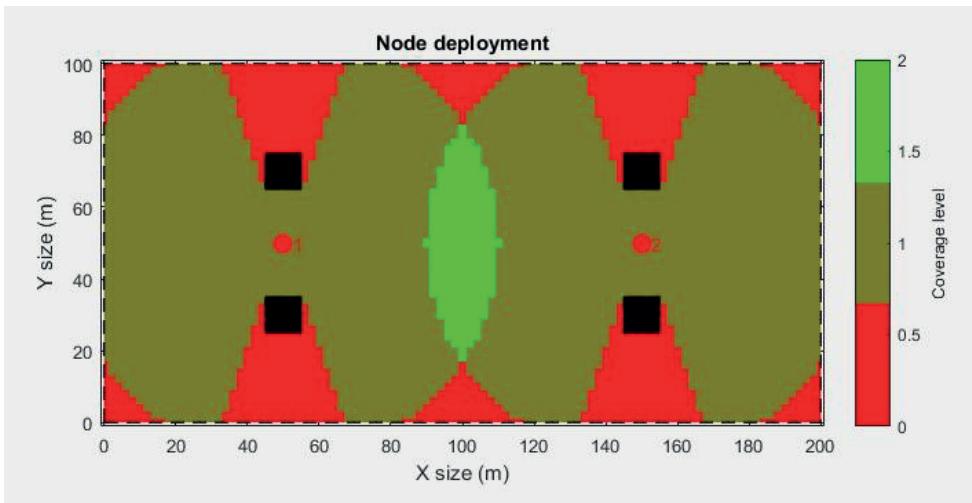


Figure 23: Sensing coverage map.

NodeId from the PlantDescription or by storing a specific identifier, such as a public key from a certificate, in the Configuration system beforehand.

- Each planned service interaction between systems should be stored as service links between the system nodes in the PlantDescription.
- A user interface available during deployment should be able to navigate the PlantDescription and transfer the previously mentioned identifier to or from the device through a local interface.

Deployment procedure:

1. Device is physically connected to the network and turned on.
2. Device connects to the network using DHCP, or a similar standardized technology applicable to the network in question
3. Device looks for the Arrowhead core systems [23] at predefined locations (e.g., on the local network or a cloud service hosted by the device supplier)
4. Core systems authenticate device as factory configured device with basic authorisation
5. The user interface is used to associate the physical device with the device node in the PlantDescription
6. The device registers with mandatory core systems before access is granted to support core systems

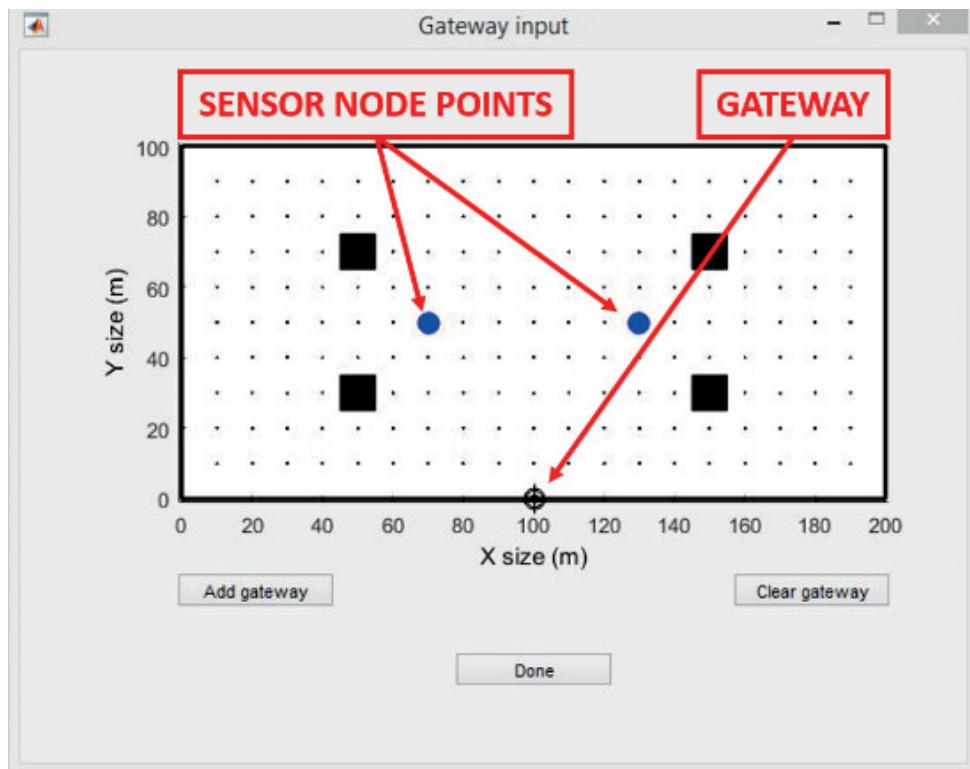


Figure 24: Gateway input tool

7. The device uses the identifier to access the Configuration store and retrieve its configuration.
8. The device configures itself to host systems according to the configuration, and the systems registers with the SystemRegistry.
9. Depending on the implementation of Orchestration used, one of the following interactions is initiated:
 - a) The SystemRegistry accesses the PlantDescription providing the association between the SystemId and the NodeId. The PlantDescription system checks for all service links to the associated NodeId and requests SystemId, for all NodeIds connected by service links. Using these provided SystemIds the Plant-Description system creates orchestration rules for all affected systems
 - b) The SystemRegistry notifies the Orchestration system with the newly registered SystemId and associated NodeId. The Orchestration system then queries

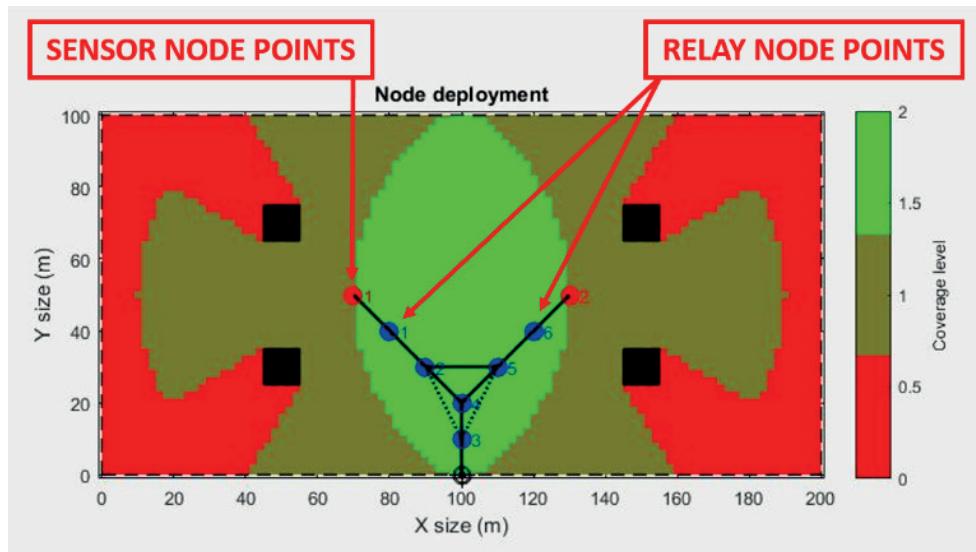


Figure 25: Relay node deployment

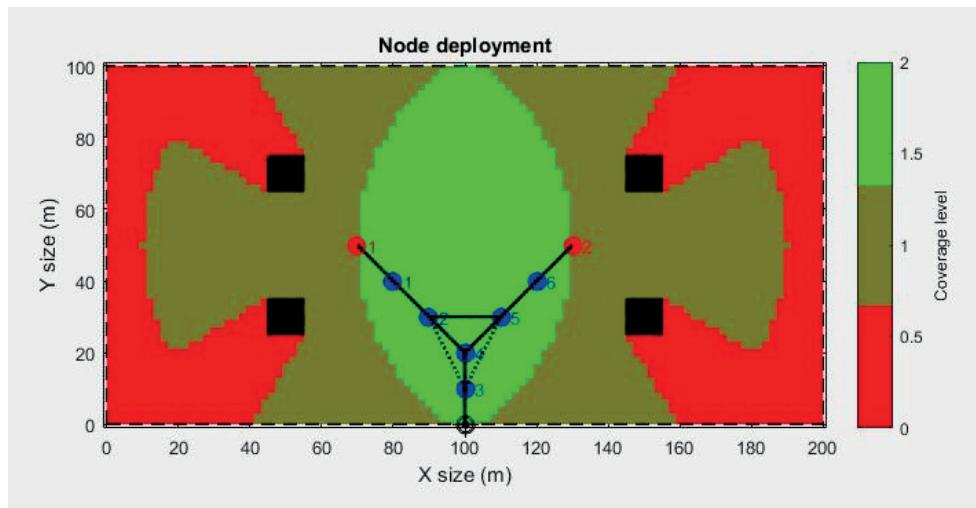


Figure 26: Initial node deployment (minimum number of nodes).

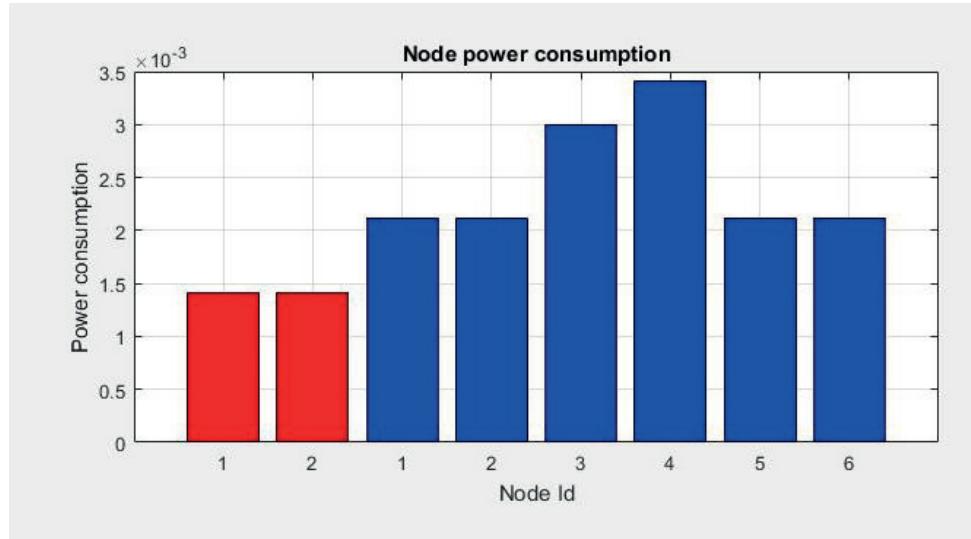


Figure 27: Initial deployment's power consumption.

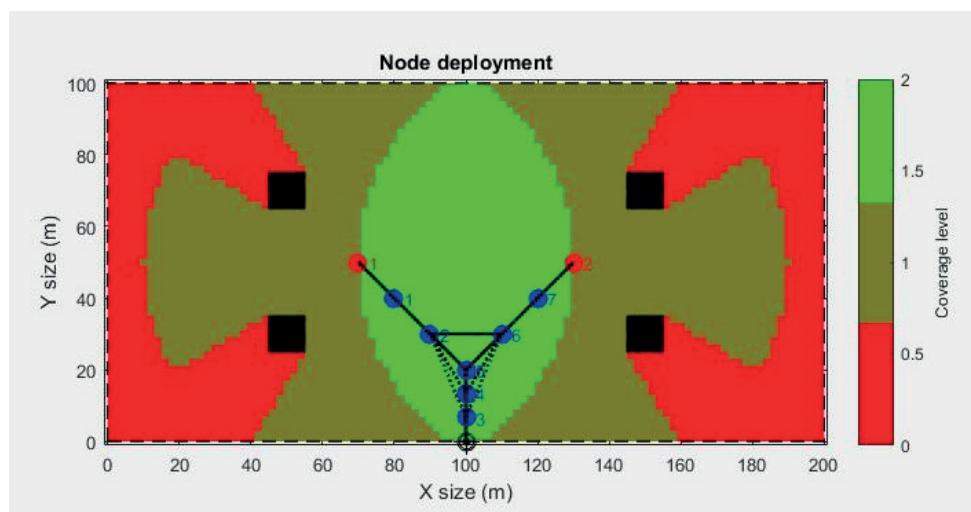


Figure 28: Node redeployment (one node added).

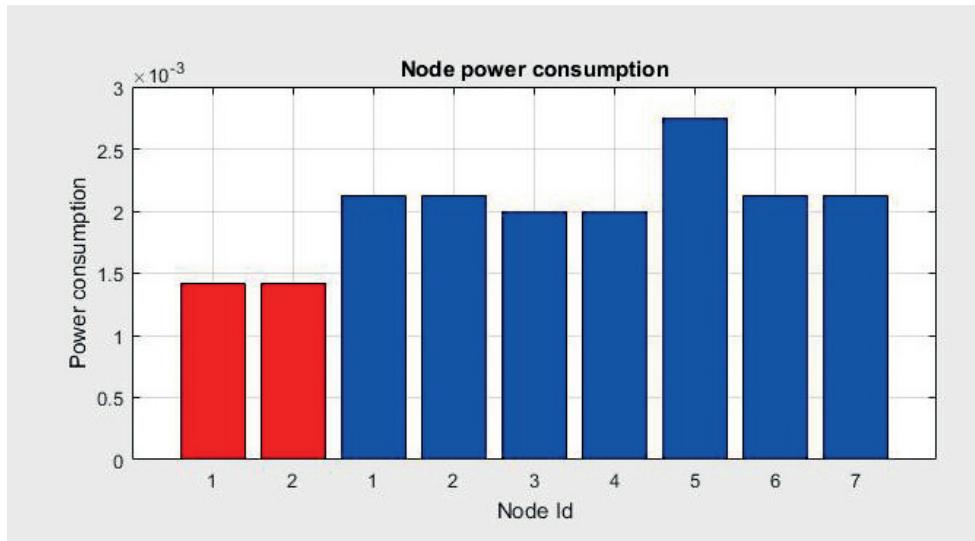


Figure 29: Redeployment's power consumption.

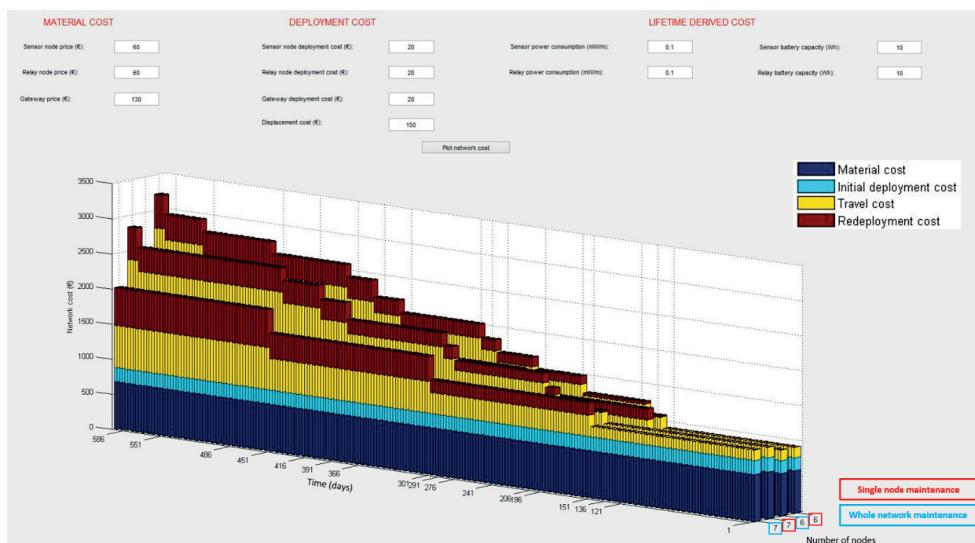


Figure 30: WSN cost tool.

the PlantDescription for system nodes linked to the NodeId by service links. It then uses the provided NodeIds to query the SystemRegistry for associated SystemIds and creates the desired orchestration rules, possibly also using other available information such as QoS or similar

- c) The newly introduced system requests an orchestration from the Orchestration system, providing either SystemId or its NodeId. The Orchestration system in turn accesses the PlantDescription (if necessary after getting the NodeId from the SystemRegistry) which responds with the appropriate service links. Using the service links, and other available information, the Orchestration system provides the requested orchestration back to the newly introduced system.

At the end of the procedure, the operators and engineers should be informed the procedure has been successful or if some part of it failed. This can be done through an event or subscription mechanism, where involved user interfaces are notified depending on the result. For example, a deployment failure could first be sent to the user interface that was used in the deployment process, and only escalated to other users if the user that tried to deploy the device is unable to solve the issue.

5.3 Scenario: PLC device monitoring

On PLC controlled industrial equipment, services are likely to be provided not by the device itself for obvious reasons related to security and production constraints, but by a higher-level network node to which the PLC device is connected. OPC/OPC UA is a widely accepted protocol for interfacing with PLC controllers and was used in the Arrowhead project to develop a tool set used to support automation system life cycle (i.e., design, deployment, operation monitoring, and maintenance). The OPC UA server is the interface to PLC devices and is used to produce/consume services that cannot be deployed at device levels.

Two main scenarios of the automation life cycle can be identified during or after the commissioning phase:

- System configuration
- System monitoring

The configuration of the PLC control system requires downloading of PLC code to the target controller. For safety, security, and practical reasons, this is typically achieved through specific and/or proprietary software using a direct connection to the PLC device, and not via web service provided or consumed by the device itself. The provision of web service is implemented at a higher level of the system architecture.

Configuration management

For PLC devices which code is generated using the CCE Mapper tools [23], the configuration data for a PLC consists of

- PLC control code
- Component to FB (Function Blocks) mapping
- Function block to I/O mapping
- Variable list (that needs to be monitored)
- PLC memory addresses/OPC UA variable mapping
- OPC UA server IP (for configuring OPC UA client)
- Service provider server IP (for configuring OPC UA client)

Configuration data generated by the Control Auto-generation tool can be versioned and stored on the application server for later retrieval by the mapper tool. The configuration data is used by the mapper tool to configure the PLC device (i.e., download of the control code) and configure the OPC UA server and OPC UA client. Once the OPC UA client is configured, it connects to both the OPC UA server and application server. The system is then ready to collect data.

The PLC generated data (e.g., state change, fault) are collected by the OPC UA server. The OPC UA client polls or is notified of PLC events and pushes the data to the application server, where the data is stored. End user applications or higher level application servers can consume and process the data.

Example of services provided by the application server layer and related to this operation and maintenance phase related scenarios are listed below:

- StoreConfig: allows the deployment tool to store a new OPC UA server configuration (i.e., OPC UA variable to PLC memory address mapping)
- GetConfig (SysId, Version): allows retrieval of a specific version of configuration data for a specific system
- GetSystemList: returns a list of systems for which configuration data is stored
- GetLog(SysId, VarTyp, start_{date}, end_{date}): Gets log of variable change for a given system, for a given variable type and time interval

5.4 Scenario: Replacement of device

For a device containing its own configuration, program code, or other local customisation, the procedure for replacement becomes much more complex than that of simply replacing a generic device that always operates in the same mode. If the component that has failed is not capable of providing a copy of its data after the failure, it is very important that up to date documentation and backup of data is readily available.

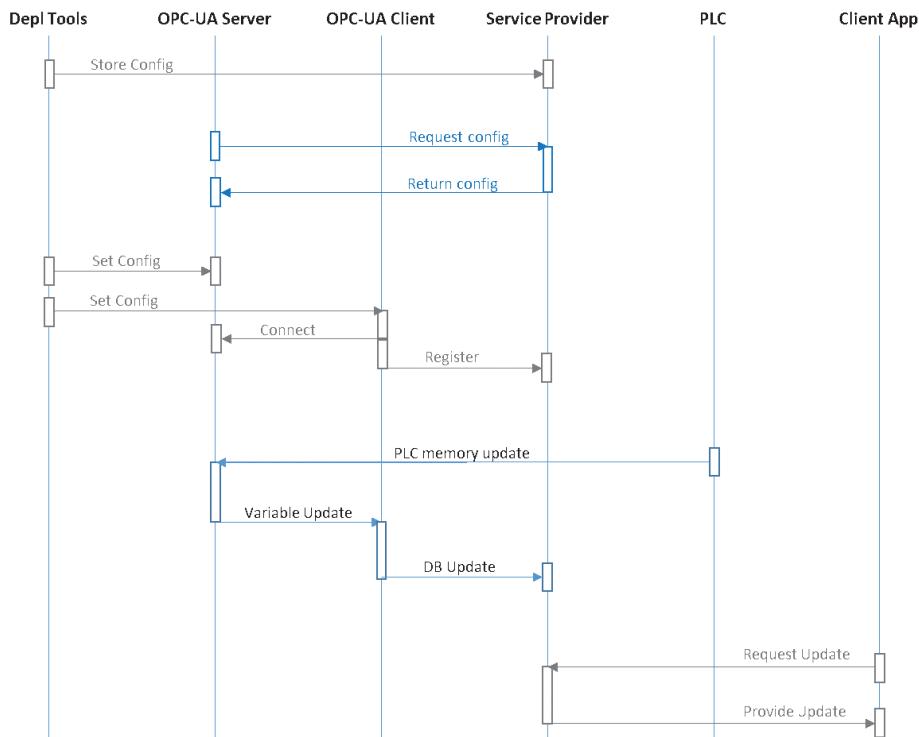


Figure 31: Retrieval of OPC UA client server connection data to monitor PLC device states.

Common current procedure

The current procedure is that all forms of configurations, code and customisations are documented and stored either electronically or on paper in a repository or archive, managed by the site owner, system supplier or a system integrator. In the event of replacement of such a device, a new device is acquired and the party responsible for maintaining the archive or repository configures the device before it is tested and installed at the site where the functionality is once again verified.

Recommended procedure for Arrowhead

As all Arrowhead devices are expected to be interoperable, it is recommended that they can support some method of retrieving all device-specific data and that this data can be used to easily deploy a replacement device.

The proposed method for this is for the relevant data to be stored at a network-accessible location, with backups, redundancies, and security measures as deemed necessary. As a replacement device is connected and authorised, the stored data can then

be accessed directly and assigned to a new device through a few services, requiring less engineering effort.

This method of replacement is intended to be enabled by the Configuration system and PlantDescription system. Further support is given through the deployment procedure that is to be used to give a newly connected device its initial network access and authorisation so that it is able to find the systems providing Configuration services.

The Configuration services are intended to provide the functionality for storing and retrieving the device-specific data, for initial configuration of systems as well as reconfiguration and replacement of devices.

The PlantDescription services are intended to give a basic common understanding of the layout of the plant or site, providing possibilities for actors with different interests and viewpoints to access their view of the same dataset. In the case of device replacement, this is useful for the technician replacing the device to assign which position the new device is in, e.g., which old device it is meant to replace.

Specific cases - Device configuration

In some cases and for some types of devices (e.g., many PLC controllers), the services required to achieve device configuration cannot be accessed by the device itself (e.g. no direct access to the network of services, limited resources, security/safety reasons, etc.). In this case the services required to achieve device configuration are consumed by an external configuration device (e.g., Laptop, machine HMI) physically connected to the PLC.

Specific cases - Device hardware versions/types

A given PLC control device might be replaced with a device from another vendor. A typical scenario is update of control systems on legacy production equipment. From an Arrowhead/Engineering scenario perspective, this implies that a given device configuration may exist in the device configuration database in several formats. In such a case, the Configuration service would therefore have to provide the ability to select configuration format as well as configuration version.

Specific cases - Device configuration formats

Device configuration might exist in various formats throughout the device life cycle. During virtual design, and virtual validation, device configuration may be generically described (e.g., XML-based description of PLC control logic). The generic description will at some point be translated into device/vendor specific format (e.g., Siemens Step 7 files) and compiled during installation on the target hardware. The Configuration service could therefore provide the ability to retrieve a given device configuration in several formats, depending on the end user requirements (e.g., update of configuration on the device itself, or device configuration design modification). Possibly services related to checking consistency between the same device configuration in various formats would

also be necessary in order to support a scenario such as the one described in Section 5.5. The need for device configuration format translation could also be investigated.

5.5 Scenario: Device configuration upload

This scenario is based on a common but not always suitable practice typically encountered in industry, which results in the device configuration being tweaked by shop floor technicians and engineers either because the initial device design configuration was flawed or because the machines physical configuration was changed and therefore the control configuration needs to be changed accordingly. In such case the software/configuration installed on the physical device may become the gold standard and should be stored in the configuration database as a new update/version of the previous configuration. The Arrowhead configuration service should therefore provide the ability to allow a device (or any device configuration system) to upload a new or update a version of the configuration in the configuration database.

References

- [1] R. Yang, *Process Plant Lifecycle Information Management*. AuthorHouse, 2009. [Online]. Available: <https://books.google.se/books?id=zmgzdjf1GR0C>
- [2] A. J. Braaksma, W. W. Klingenberg, and P. P. van Exel, “A review of the use of asset information standards for collaboration in the process industry,” *Computers in Industry*, vol. 62, no. 3, pp. 337 – 350, 2011. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0166361510001387>
- [3] IEC 62424 *Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools*, International Electrotechnical Commission Std.
- [4] R. Drath, A. Luder, J. Peschke, and L. Hundt, “Automationml - the glue for seamless automation engineering,” in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, Sept 2008, pp. 616–623.
- [5] MIMOSA OSA-EAI - *Open System Architecture for Enterprise Application Integration*, MIMOSA OSA-EAI Std. [Online]. Available: <http://www.mimosa.org/mimosa-osa-eai>
- [6] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer, 2009.
- [7] V. Vyatkin, *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*, third edition ed. International Society of Automation, 2016.
- [8] ISO/IEC Std.

- [9] *ISO 15926 Industrial automation systems and integration - Integration of life-cycle data for process plants including oil and gas production facilities*, International Organization for Standardization Std.
- [10] *fastAPI - Standard för fastighetskommunikation*, SABO Std. [Online]. Available: <http://www.fastapi.se/>
- [11] *IEC 61850: Power Utility Automation*, International Electrotechnical Commission Std.
- [12] M. Schleipen. (2014) Open standards for Industry 4.0 - Tools and offer around AutomationML and OPC UA. [Online]. Available: http://www.iosb.fraunhofer.de/servlet/is/46944/AutomationML_en.pdf
- [13] A. T. Johnston, “OpenO&M and ISO 15926 Collaborative Deployment,” October 2009. [Online]. Available: <http://www.mimosa.org/presentations/openom-and-iso-15926-collaborative-deployment>
- [14] P. Adolphs, H. Bedenbender, D. Dirzus, M. Ehlich, U. Epple, M. Hankel, R. Heidel, M. Hoffmeister, H. Huhle, B. Kürcher, H. Koziolek, R. Pichler, S. Pollmeier, A. Walter, B. Waser, and M. Wollschlaeger, “Status Report - Reference Architecture Model Industrie 4.0 (RAMI4.0),” VDI - Verein Deutscher Ingenieure e.V. and ZVEI - German Electrical and Electronic Manufacturers’ Association, Tech. Rep., July 2015. [Online]. Available: <http://www.zvei.org/Downloads/Automation/5305PublikationGMAStatusReportZVEIReferenceArchitectureModel.pdf>
- [15] H. Kagermann, W. Wahlster, and J. Helbig, “Recommendations for implementing the strategic initiative industrie 4.0,” acatech - National Academy of Science and Engineering, Tech. Rep., 2013.
- [16] S. Plosz, M. Tauber, and P. Varga, “Information Assurance System in the Arrowhead Project,” *ERCIM News*, vol. 97, p. 29, April 2014, iSSN 0926-4981.
- [17] *ISO/IEC 27005 , Information technology - Security techniques - Information security risk management*, International Organization for Standardization Std., 2011.
- [18] M. Howard and D. E. Leblanc, *Writing Secure Code*, 2nd ed. Redmond, WA, USA: Microsoft Press, 2002.
- [19] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, “Uncover security design flaws using the STRIDE approach,” *MSDN Magazine*, Nov. 2006. [Online]. Available: <http://msdn.microsoft.com/en-us/magazine/cc163519.aspx>
- [20] *Telecommunications and internet protocol harmonization over networks (TIPHON) release 4; protocol framework definition; methods and protocols for security; part 1: Threat analysis*, ETSI Technical Specification, 2003.

- [21] M. Barbeau, "Wireless security in the home and office environment," Technical Reports, Carlton University, Tech. Rep., 2010.
- [22] C. Schmittner, T. Gruber, P. Puschner, and E. Schoitsch, *Computer Safety, Reliability, and Security: 33rd International Conference, SAFECOMP 2014, Florence, Italy, September 10-12, 2014. Proceedings*. Cham: Springer International Publishing, 2014, ch. Security Application of Failure Mode and Effect Analysis (FMEA), pp. 310–325. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10506-2_21
- [23] X. Kong, B. Ahmad, R. Harrison, A. Jain, Y. Park, and L. J. Lee, "Realising the open virtual commissioning of modular automation systems." in *Proceedings 7th International Conference on Digital Enterprise Technology*, Athens, Greece, 2011, pp. 1–9.

