FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Service Oriented Manufacturing Infrastructure

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

M.Sc. B.Sc. Ahmed Ismail
Registration Number 1429613

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof.Dr. Wolfgang Kastner

The dissertation has been reviewed by:

Wolfgang Kastner	Paul Pop	Dirk Timmermann

Vienna, 24th January, 2018

Ahmed Ismail

Erklärung zur Verfassung der Arbeit

M.Sc. B.Sc. Ahmed Ismail
Nordportalstraße 02, 1020 Vienna, Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 24. Jänner 2018

Ahmed Ismail

Dedicated to my parents and sisters.

Acknowledgements

First and foremost, I thank my professor, Wolfgang Kastner, for his endless support while supervising my dissertation. His knowledge, dedication, humility, and genuineness as a person sets a high standard for me to live up to. I'd also like to thank Guenther Gridling for the many long and helpful technical and academic discussions. I thank Ethar Ismail and Raisa Trubko for their time spent proof reading my work. I'd like to thank my examination committee for taking the time to review this dissertation. I also thank all the professors of the Doctoral College Cyber-Physical Production Systems, of my previous courses, and the AutomationML and EDBT'17 summer schools. A special thanks to all my lab mates, past and present, for their cheerful openness and motivating conversations and for making Austria feel just like home. Finally, I thank my family for accepting the many years of distance in the pursuit of knowledge.

This dissertation is supported by TU Wien research funds as part of the Doctoral College Cyber-Physical Production Systems.

Abstract

This dissertation is concerned with the design and implementation of infrastructural systems for resilient Machine-to-Machine (M2M) communication in distributed Cyber-Physical Production Systems (CPPS). For this purpose, a number of technologies are selected and applied in congruence with the principles of Smart Manufacturing. Thus, this dissertation investigates the use of Service Oriented Architectures (SOA), M2M communication middleware systems, overlay networking solutions, and other technologies to improve the agility, resilience, and interoperability of manufacturing infrastructure.

First, the concept of SOAs and how they may be applied in current enterprises to achieve flexibility, agility and interoperability is addressed. As such, the technical state of current industrial enterprises and the characteristics of the Service Oriented (SO) approach are detailed to highlight the competitive advantage possible through service orientation. A review of preliminary SO Reference Architectures (RA) delivered by major European Union (EU) research projects is conducted to determine their features and possible shortcomings. Realisations of the architectures are also discussed to underline their choices in technologies and their delivered technical innovations. Based on the findings of the review, the SO Open Platform Communications Unified Architecture (OPC UA) is selected as the base technology for the envisioned system.

Following a bottom-up approach to system development, this dissertation proceeds to investigate the rigid networking infrastructure in manufacturing enterprises. It evaluates the possibility of using Peer-to-Peer (P2P) networking technologies to create a cohesive, fault-tolerant network of components for the non-real time management, replication, storage, and sharing of plant data and service components. A cooperative P2P overlay network is proposed as the most applicable architecture for manufacturing systems. While the overlay network proposed may be used as a transport layer for OPC UA, it is in fact developed as a middleware-agnostic protocol, thus affording it wider applicability in the domain. The design is also evaluated through a prototypical implementation that demonstrates the viability of the approach.

Finally, the OPC UA specifications are reviewed to highlight possible items for enhancement. Specifically, two proposals are evaluated. The first assesses the practicality of employing a dedicated service for the distributed coordination of redundant OPC UA servers. The second applies a queueing service to shield resource-constrained OPC UA servers from high rates of concurrent asynchronous service calls (SC) that may lead to resource exhaustion. Both proposals are evaluated programmatically and the code is open sourced. Results demonstrate the feasibility of their respective designs.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Several passages in this chapter are reproduced verbatim from the following publication:

1. Ahmed Ismail and Wolfgang Kastner. Service-Oriented Architectures for Interoperability in Industrial Enterprises. In Stefan Biffl, Detlef Gerhard and Arndt Luder (eds.). Multi-Disciplinary Engineering for Cyber-Physical Production Systems. Springer International Publishing AG, May 2017.

2. Ahmed Ismail and Wolfgang Kastner. Vertical Integration in Industrial Enterprises and Distributed Middleware. International Journal of Internet Protocol Technology 9(2/3):7989, 2016.

In a similar fashion to how the Internet redefined the business-to-consumer industry, avid efforts are being applied to bring about revolutionary changes to the manufacturing sector using new emerging technologies [1]. This revolutionary movement is currently being spearheaded under the conceptual term of the 'Internet of Things' (IoT). This term was coined by technologist Kevin Ashton in 1999 when he claimed that with ubiquitous sensing and autonomous data collection technologies becoming a reality, the Internet was shifting from connecting humans to connecting devices, hence the term the 'Internet of Things' [2].

Since then, IoT research tracks have become well-established academic fields in the particular branches of building and home automation, transportation, and energy sectors. As of recently, the IoT movement has set its sights on production systems to achieve the vision of a fourth industrial revolution. Thus, several national and trans-national initiatives, some of which are listed in Table 1.1, have been created over the past years to investigate the application of transformative technologies to the manufacturing industry. Their goal is to maximise the economic competitiveness of their respective manufacturing sectors through current CPPS. These CPPS consist of computationally controlled physical elements. Thus, they present a platform through which strategies from Information Technology (IT) can be applied to enhance the sustainable competitiveness of manufacturing enterprises.

The appropriate selection and deployment of technologies in manufacturing is typically done according to a manufacturing strategy. A manufacturing strategy, according to [10], is a framework for the design, organisation, management, and development of

Initiative	Countries	Description
Industrial Internet Consortium	243 Member Organisations	Accelerator for the development and adoption of secure connected and controlled machines and devices [3]
Industrie 4.0	Germany	National strategy for the digitisation and integration of the full manufacturing value chain [4]
Intelligent Manufacturing Systems	EU, Mexico, South Africa, United States of America (U.S.A)	Industry-led international collaboration for research and development (R&D) and industrial deployment in advanced manufacturing [5]
Production of the Future	Austria	R&D Program for joint industry and research efforts in the applied sciences for the development of competitive products in the Austrian manufacturing sector [6]
National Network of Manufacturing Institutes	U.S.A	Joint network of industrial, academic, and governmental partners for the promotion of R&D in the advanced manufacturing sector [7]
Smart Industry	Sweden	National strategy for renewed industrialisation through focused efforts in a national Industry 4.0, sustainable production, industrial skills boost, and a test bed Sweden for lead research in manufacturing production [8]
Smart Manufacturing Leadership Coalition	U.S.A	Developing an open platform for the inter-organisational collaboration of networked industrial applications to enable the proliferation of Smart Manufacturing in enterprises [9]

Table 1.1: National and trans-national initiatives for the advancement of manufacturing technologies.

a manufacturing enterprise's resources. It is used to focus the manufacturing decisions of a company towards achieving a select number of characteristics with the purpose of continuously improving the company's competitive advantage. The next section discusses a number of manufacturing paradigms to discern the governing strategy for this dissertation.

1.1 Manufacturing Strategies

Over the years, many different manufacturing strategies have emerged. Several of them are summarised in Table 1.2. Perhaps one of the most notable paradigms is that of lean manufacturing. Born of Toyota's just-in-time system, the lean production strategy is based on the following principles described in [10]:

- employing a broadly trained workforce instead of specialised personnel
- empowering employees to find and resolve production issues

- using informal and horizontal communication instead of hierarchical structures
- focusing on "production throughput flow" in place of resource utilisation
- employing on-demand production flows instead of those formally dictated through centralised scheduling
- utilising a product-based rather than a process-based layout
- exercising zero-tolerance towards any manufacturing defects
- centralising waste elimination and continuous improvement initiatives
- considering inventory as waste
- reducing set-up times
- establishing long-lasting cooperative relationships with suppliers in place of adversarial ones
- carrying out multiple product development activities at the same time using cross-functional teams

Manufacturing Strategy	Description
Focused Manufacturing	Specifies the functions of each part of a manufacturing system with specific products, technologies, volumes, and markets to limit and consequently improve its ability to achieve and excel in performance [10]
Lean Manufacturing	Focused on the elimination of waste through increased flexibility and monitoring and corrected metrics [10]
Agile Manufacturing	Quick response to regular and unpredictable changes through control and mitigation strategies [10]
Flexible Manufacturing	Adopts computerised control systems to allow for quick changes to production [11]
Sustainable Manufacturing	Focused on the management of scarce resources, and the sustainability of products, production systems, and process [12]
Digital Manufacturing	Uses information technology systems to minimise product development times and cost, increase product customisation and quality, and improve enterprise response times to the market [13]
Cloud Manufacturing	Centred on the use of service oriented architectures and cloud computing technologies [14]
Intelligent (Cognitive) Manufacturing	Using systems that share traits and principles with complex biological systems to supplement or replace human problem-solving in manufacturing [15]
Holonic Manufacturing	Adopts a hierarchy of self-reliant and autonomous agents for process control to improve the ability of manufacturing systems to adapt to product evolution and improve performance in out-of-bounds operating conditions [16]
Smart Manufacturing	Focused on the optimised application of resources and the workforce to achieve the on-time production of high quality goods while maintaining the enterprise characteristics necessary for the company to control and respond to internal and external stimuli [14]
Scalable Manufacturing	Development and implementation of algorithms for the (elastic) scalability of different types of manufacturing systems for reductions in manufacturing costs and waste and improvements in production throughput [17]

Table 1.2: A subset of notable manufacturing strategies.

Lean manufacturing quickly allowed the Japanese manufacturing industry to consolidate a substantial portion of the world's export market in the 70's through to the 90's. However, with the rise of globalisation and the rapid rates of technology development in the 90's, lean manufacturing soon gave way to agile manufacturing, a strategy focused on proactively adapting to frequent and unforeseen internal and external changes by controlling and dealing with the effects of those changes. Examples of change control given in [10] include:

- monitoring and forecasting the change
- confining the effect of changes to specific equipment
- outsourcing tasks that experience drastic changes to other firms
- instilling redundancy to stabilise the system in the face of change or substituting the source of change with another
- negotiating with the customers to reduce changes in customer demands
- using advertising and promotions to reduce the elasticity of market volume and rate demands
- employing preventative maintenance and staff training to reduce the variability in equipment availability and personnel behaviour

Handling unpredictable change, however, is dependent on the flexibility of the product, mix, volume, delivery, and system robustness. This is based on the definition of flexibility as the range of states available and the cost associated with the transitioning between the different states.

In recent years, trends have shifted towards smart manufacturing, which is a current paradigm that evolved from lean and agile principles. According to [14], smart manufacturing is focused on the optimised application of resources and the workforce to achieve the on-time production of high quality goods while maintaining the enterprise characteristics necessary for the company to control and respond to internal and external stimuli. Smart manufacturing, however, places special emphasis on the role of emerging technologies. Thus, it calls for the digitisation of all manufacturing-relevant activities as well as the adoption of technologies such as advanced sensors, SOA, and big data analysis to achieve a competitive market advantage.

It is, therefore, the case that one of the largest opportunities in manufacturing currently lies in achieving the still-relevant goals of lean and agile manufacturing through a technology-centric smart manufacturing approach. This involves employing technology-driven solutions for the definition of a characteristic enterprise with "easy access to integrated data whether it is customer driven, supplier driven, or product and process driven", "modular production facilities that can be organised into ever-changing manufacturing nodes", and "data that is rapidly changed into information [for the expansion of] knowledge", amongst other things [18].

The pursuit of these characteristics requires an understanding of the current technical landscape of manufacturing enterprises. This context is defined in the next section.

1.2 Technical Features of the Industrial Enterprise

A reference architecture is a structured meta-model representing the various functional elements and interactions of an enterprise system. Industrial enterprises use a reference architecture in order to allow for the rapid generation of a useful system architecture that adopts all of the relevant insights and best practices gained from years of previous deployments. Developing for manufacturing systems without taking into account the restrictions imposed by existing architectures risks the breaking of application dependencies vital to the manufacturing process. This section focuses predominantly on the Purdue Enterprise Reference Architecture (PERA) framework as it is widely accepted by industry and is compatible with multiple manufacturing standards, such as ISA 95, ISA 88, and IEC 62443 [19, 20, 21].

Using the ISA 95 and ISA 88 models, PERA essentially segregates the elements comprising an industrial enterprise system into separate zones and conduits. The system is essentially divided into 5 functional layers, as shown in Fig. 1.1, that correspond to the 5 layers of the automation pyramid (see Fig. 1.2). The lowest layer, level 0, is the actual physical process. Level 1 consists of the device communication networks directly in control of the physical processes. The second layer comprises the control and automation network. This level can directly access the process and discrete devices of level 1 to set or reconfigure them as needed. Level 3 consists of operations management systems, such as the Manufacturing Execution System (MES). Level 3 components may only read from layers 1 and 2. Level 4 is where the enterprise system is located and is where the bulk of the business process network exists. The application of the PERA model has been extended with time and currently includes a sixth layer and a Demilitarised Zone (DMZ). The sixth layer, level 5, is where centralised IT systems and their associated functions are situated. The DMZ, on the other hand, is in place to manage access from levels 4 and 5 to the data and network of levels 0 - 3. The recommended practice is to have no traffic cross the DMZ. Instead all traffic should either originate or terminate within the DMZ. Therefore, data sharing and application servers are normally found in the DMZ [20, 21, 22].

This kind of segmentation is done to increase the manageability and security of the enterprise. However, further impositions on the enterprise exist due to the physical and logical constraints of the enterprise's assets. Physically, devices may be connected using legacy serial interfaces (e.g. EIA-232/422/485), fieldbus systems, and wired and wireless Ethernet and IP-based technologies. Although devices with both interface types may be used to physically bridge the two networks together, the protocols may have different demands in bandwidth, latency, and other communication-related requirements. At the messaging layer, these protocols may also differ in their message formats and exchange mechanisms, as well as in other features. The process of bridging together these various systems requires special devices and techniques that are designed and implemented carefully to safely allow them to share data [24, 25].

By convention, there are two approaches for managing this heterogeneity and technical complexity, namely tunnelling and translation. Tunnelling involves the use of routers

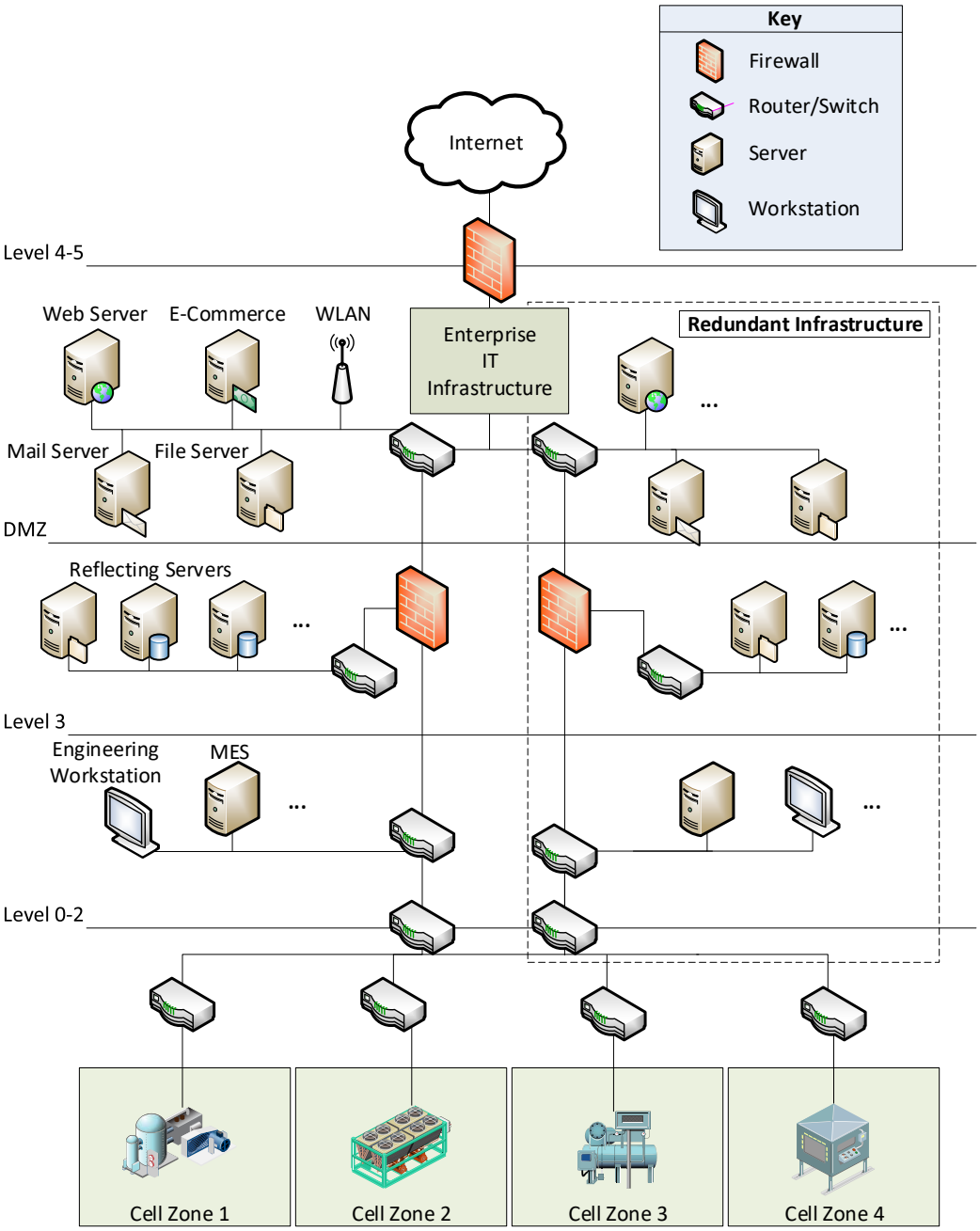

Figure 1.1: A typical manufacturing enterprise network architecture.

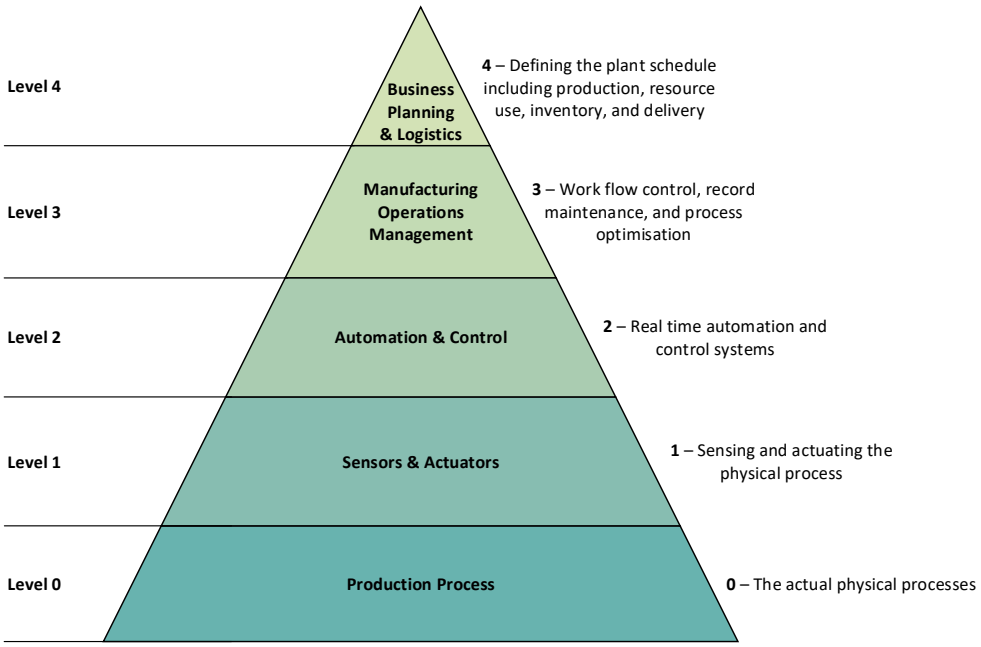

Figure 1.2: The five layers of the automation pyramid [23].

for the encapsulation of one protocol's data inside the payload of another and treating the channel as a transparent communication medium. Translation uses gateways as intermediaries to carry out data mappings on behalf of communicating devices to allow them to exchange information using their native protocols. Each of these methods has its own drawbacks in relation to capabilities and implementation complexity, and is often considered to be costly in engineering efforts [26].

1.3 Problem Statements & Hypotheses

Problem statement 1 Enterprise processes are compounded by their technological choices and implementations resulting in heterogeneous infrastructure that introduces stiff resistance to change.

The main vision is to establish flexible, agile, and resilient distributed systems in manufacturing enterprises. This involves overcoming the complexities of manufacturing infrastructure which typically use a large number of protocols, are governed by numerous standards, and are present in many various architectural forms. This heterogeneity proves to be a major hindrance to the adoption of modern technologies that are increasingly dependent on the easy accessibility of data and devices. The resulting systems normally include many dependencies (for example, to a technology stack) that make

rapid changes to the organisation extremely difficult. Thus, the enterprise cannot be described as agile or flexible. The aim of this dissertation is to counter this heterogeneity and inflexibility by enhancing data exchange capabilities and allowing an organisation to quickly adapt to changing conditions and create a competitive advantage for itself [27].

Hypothesis 1 Given the complex and heterogeneous technical landscape in manufacturing, a SOA would provide a suitable pathway for the pursuit of agile characteristics in modern enterprises.

From a technical perspective, Chapter 2 will demonstrate that a SOA is the most appropriate approach in the pursuit of agile characteristics. This is a field that is concerned with the creation of modular IT and production systems that enhance an enterprise's capabilities for information exchange, technological independence, and component reuse. The result would effectively be an industrial environment of operational flexibility and responsiveness [28].

Problem statement 2 The proliferation of SOA in the manufacturing domain complicates the process of selecting an appropriate RA for the development of SO manufacturing infrastructure.

The properties and benefits of the SO approach have led several research projects to pursue and outline RAs for highly interoperable industrial environments based on SOA. These efforts have been extensively documented to facilitate future system implementations. However, the proliferation of RAs complicates the decision making process with regards to technology adoption.

Hypothesis 2 In contrast to the recently proposed research-driven SO RAs, the OPC UA standard currently provides a mature, standardised, well-adopted, and well-supported SO solution for the integration of M2M communication infrastructure.

An analysis of several prominent RAs is given in Chapter 2 to determine their applicability to contemporary manufacturing enterprises. Chapter 2 will show that these research-driven RAs are vulnerable to criticisms from stakeholders and low adoption rates. Instead, the OPC UA specifications family is seen as a suitable alternative for the implementation of interoperable SO M2M communication infrastructure.

The proper adoption of the SO architectural pattern should allow for the development of an integrated system for the deployment of meaningful services. The envisioned system consists of pervasively deployed, dynamic, robust, versatile, and extensible services for the agile execution of cyber physical production processes. Given the safety critical nature of manufacturing operations, this first and foremost implies designing a system that is resilient to failure at the communication and application layer. This translates to a system with integrated traffic engineering and self-preservation measures that allow it to sustain business operations in the presence of failure by continuing to serve essential services. A diagram demonstrating this envisioned system is shown in Fig. 1.3.

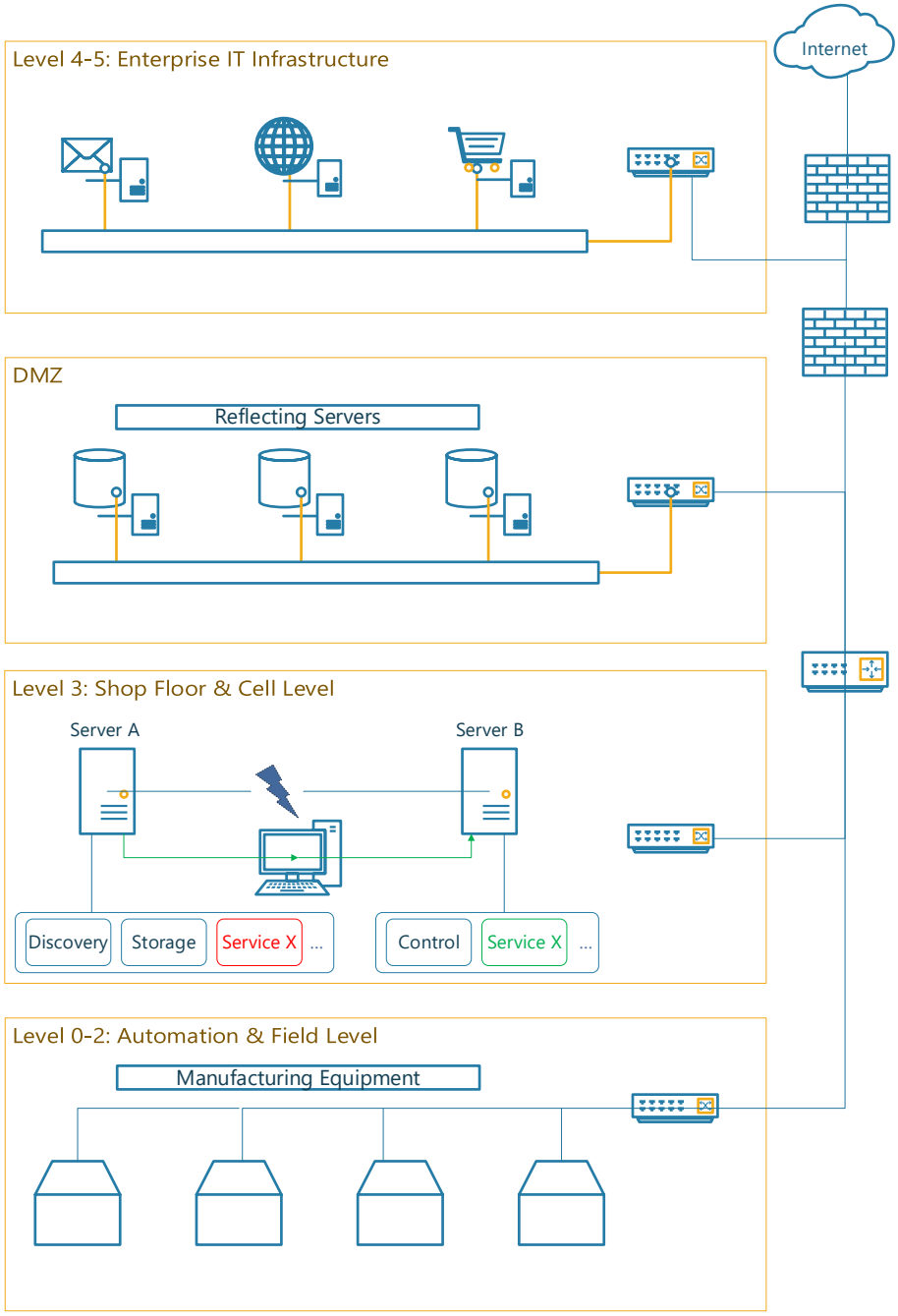

Figure 1.3: The envisioned resilient and robust SO manufacturing infrastructure. Service X on server A fails and is replicated on a different server (server B). With the link to the new service down, requests are re-routed through existing infrastructure to establish a new path for the continuity of data exchange.

Problem statement 3 SO CPPS require flexible, scalable, and failure-resistant transport protocols for a dynamic system of services. This is in contrast to current technologies which often limit these properties.

Following a bottom-up approach, the first task to be tackled is the design and development of a flexible, scalable, and failure-resistant communication protocol. This is in contrast to the dominant solutions in manufacturing which are often times found to be rigid in structure, have limits on the maximum number of devices, include a Single Point of Failure (SPoF) in their respective architectures, and require special hardware for their operation [29].

Hypothesis 3 The domain of P2P networks as cooperative systems has suitable properties for the design of a failure-resistant protocol for SO manufacturing infrastructure. Deviations from their typical nature will be necessary to adapt them to the manufacturing domain.

Developing a network for resilient communication atop of existing infrastructure designates it as an overlay network. Of the various types of overlay networks possible, P2P networks happen to be one of the most well-developed and understood. Chapter 3 will demonstrate that P2P solutions are highly compatible with SO systems. However, P2P technologies are themselves classified based on their degree of centralisation, hierarchicalness, and other features that have direct implications on their respective architectures and capabilities. A specific subclass of P2P technologies known as P2P networks as cooperative systems is expected to be the most appropriate solution for operating in a manufacturing landscape. This is because it is a subclass concerned with the bridging, merging, and sharing of resources between disjoint systems of machines. Thus, it is compatible with the architectural constraints of manufacturing infrastructure, which, as previously stated in Section 1.2, is governed by numerous restrictions that may limit the flow of data between different groups of machines. Yet, since designs from the subdomain of P2P networks as systems are typically developed for use on the open nternet, it is also likely that modifications to their characteristic features will be needed to adapt them to the localised nature of manufacturing infrastructure.

Problem statement 4 OPC UA based distributed systems have coordination needs for the safe operation of redundant servers that would require an extensive investment on the part of developers.

The second task involves reinforcing the application layer, which, in the envisioned system, is inexplicably linked with OPC UA. OPC UA operates using a client-server architecture to develop large distributed systems for M2M communication. Redundancy is a standard feature for the resilience of services. Although OPC UA accounts for redundancy in its specifications, it does not define the measures necessary for the coordination of redundant components. Such mechanisms are required to ensure the safe operation of simultaneously running elements. The development of coordination logic is generally considered to be a complicated and expensive task [30].

Hypothesis 4 Apache ZooKeeper provides a suitable platform for the development of a service that can meet the coordination needs of redundant OPC UA servers.

Coordination is a key ingredient of large distributed systems. It is often needed for

replication, leader election, group membership, cluster management, service discovery, resource fencing, and barrier orchestration [31]. Since coordination is a task common to many distributed applications, Yahoo! developed a popular and open source platform for coordination named Apache Zookeeper. This platform incorporates a number of primitives and a file-system like Application Programming Interface (API) specifically to reduce the effort needed in developing coordination logic. The application-agnostic Apache ZooKeeper may have the characteristics necessary to generate a coordination service that meets the requirements of redundant OPC UA servers.

Problem statement 5 The push-based client-server communication model in OPC UA leaves servers vulnerable to request overloads and the possibility of resource exhaustion.

Another issue relating to the OPC UA ecosystem concerns its client-server architecture. This approach to message exchange implies that service calls are executed via a push-based communication model. This means that clients push requests to servers which are then expected to process the service calls and possibly respond. The power to communicate is therefore with the clients. This means that OPC UA servers are vulnerable to scenarios involving high volumes of concurrent requests that may cause servers to exhaust their available resources and enter a degraded or failed state.

Hypothesis 5 A queuing service can safely install rate-throttling capabilities in an OPC UA system and circumvent the vulnerability of OPC UA servers to request overloads.

Servers may avoid resource exhaustion by limiting the number of requests that they can receive at any given time. A possible solution may use a mediator service that can throttle the flow rate of requests to each server in the system.

1.4 Goals and Methodology

The problem statements and hypotheses of the previous section may be redefined as a concise statement of sub-goals designed to achieve the envisioned system. The goals for this dissertation are summarised as follows:

1. Identify the current landscape of SOAs for manufacturing enterprises in academic literature.
2. Select an appropriate SO communication architecture for the development of the envisaged system.
3. Identify and adapt appropriate technologies and strategies to the selected SO communication architecture in support of more flexible and resilient manufacturing infrastructure.

The implemented methodology surveys recent literature relevant to the topics involved in this dissertation. The state of the art review on preliminary SOAs from academic literature leads to the selection of an appropriate architecture that forms the foundation of this dissertation. The selected architecture is analysed to discern and propose appropriate technical enhancements. The proposed changes are engineered and executed as prototypical implementations to evaluate their viability.

1.5 Dissertation in Brief

This dissertation addresses the smart-manufacturing paradigm as applied to M2M communication infrastructure. Through the use of SOAs and the implementation of versatile services for coordination, SC rate management, and failure-resistant communication, an approach is outlined for the development of more flexible, agile, and resilient distributed systems in manufacturing enterprises.

In ***Chapter 2***, the various elements involved in the development of SO solutions for CPPS are investigated. The features of SO solutions are discussed to highlight the benefits that they may deliver in alleviating the current challenges to manufacturing infrastructure. The SO RAs of five major EU research projects are surveyed to highlight their main characteristics. Realisations of these architectures are analysed to discern the compatible technologies used to guarantee system-wide interoperability. An architecture analysis framework developed by Angelov *et al.* in [32] is then applied to the five architectures to determine the fluency with which they may be translated to concrete implementations. The results show that the architectures are either over or under-specified, and are in certain cases missing critical elements needed for implementation. Previous results presented in [32] indicate that the analysed RAs are vulnerable to low adoption rates and criticisms by stakeholders. Rather than contribute to the proliferation of available SOAs through the development of another competing architecture, a mature and widely accepted SO technology is chosen for the remainder of the work presented in this dissertation. Specifically, this is the OPC UA M2M communications specifications family. The remainder of this dissertation then centres around the development of resilient middleware infrastructure that is tightly bound to this standard.

Enhancements for OPC UA are presented in ***Chapter 3*** and ***Chapter 4***. Following a bottom-up approach, ***Chapter 3*** addresses the resilience of manufacturing systems by developing an alternate transport layer based on P2P networking technologies with the goal of creating cohesive and failure-resilient communication systems. Cooperative P2P overlay networking is selected as the most appropriate technology domain for this purpose based on the requirements of manufacturing systems. This is because this field allows for the development of networks composed of multiple overlays. These can then be used for inter-system traffic engineering, inter-system content-sharing, and to create expanded systems of networks. Thus, participating infrastructure may organise itself into self-contained systems of cooperative nodes. These systems would be able to dynamically reconfigure themselves to restrict or facilitate message passing between nodes to meet the changing policies and requirements of a manufacturing enterprise. Thus, the result would be an enterprise-wide communication system that is compatible with the constraints of a manufacturing system and is resilient to both failures and network churn. To demonstrate this system, a service-oriented application is developed through an example that converts a vanilla P2P networking protocol, Chimera, into a cooperative systems protocol. The system is evaluated through a prototypical implementation in C and tested as virtual deployments on 64-bit Xen Project servers and 32-bit embedded devices. It is important to note that the resulting protocol is in fact middleware-agnostic and can

therefore be used by both OPC UA and non-OPC UA applications that share the same common need for survivable communication systems.

Chapter 4 then proceeds to address specific enhancements for the OPC UA based application layer. It begins by addressing the coordination of redundant OPC UA servers. The OPC UA specifications family may be used to build large distributed systems, such as a Supervisory Control and Data Acquisition (SCADA) system. These systems typically have several coordination requirements that make sure that the different independently and concurrently running components can operate safely. In the case of redundant OPC UA servers, this includes needs for address space synchronisation and replication, failure detection, and resource fencing. This is because OPC UA necessitates that redundant OPC UA servers expose an identical address space to all connected clients. Failure detection and automated failover measures are needed because certain redundancy modes can only allow for a specific number of active servers that are connected to downstream devices. As coordination is a task common to many distributed systems, Yahoo! developed an open source a coordination service, Apache ZooKeeper, that provides strong guarantees for consistency, ordering, and durability, and implements a number of primitives that allow for the rapid development of coordination functions. The use of such a service reduces the time and cost needed to implement and meet the coordination requirements of an application and thus allows developers to focus on the application logic instead. In summary, this chapter presents an integrated system of OPC UA and Apache ZooKeeper that meets the aforementioned coordination needs of redundant OPC UA servers. A detailed description of the architecture, data model, and components of the resulting system is given. An open source prototype is developed using the open62541 and ZooKeeper C libraries. The resulting system demonstrates its ability to provide runtime address space synchronisation, failure detection, automated failover, and contention resolution.

Next, Chapter 4 also investigates the communication mechanisms of OPC UA's client-server architecture. This communication primarily takes the form of SC and can effectively be considered remote procedure calls (RPC). Thus, OPC UA operates using client-side push-based communication. This leaves OPC UA servers open to request overloads as too many SCs may be submitted concurrently to a server in a short time span. The server may subsequently enter a failed or degraded state as it exhausts its available resources in trying to process these requests. The loss of a service in an online manufacturing system is considered to be highly undesirable and may cause extensive financial, human, and environmental losses. Measures from the standard to counter this vulnerability are found to include the use of redundant OPC UA servers and service level indicators. Effectively, these concepts amount to capacity planning and simple load balancing. Since they do not alter the push-based communication mechanism of OPC UA, the vulnerability to request processing overload persists. Thus, the use of a rate throttling service that mediates SCs on behalf of servers is proposed. This is expected to shield servers from traffic bursts that may result in unwanted consequences. Similar to the coordination service, this service is modelled on the use of Apache Zookeeper. An architecture, data model, and communication flow is detailed and an open source pro-

totype based on the open62541 and ZooKeeper C libraries is developed. The prototype also implements a safety measure that allows clients to circumvent the queuing service in case of emergencies, where service calls must be immediately processed regardless of the state of the queue.

Chapter 5 concludes by discussing the contributions of this work in the context of the problem statements and hypotheses outlined in Section 1.3. This chapter also examines on the presented system in the context of Reference Architecture Model Industrie 4.0 (RAMI 4.0). Finally, an outlook on possible future work is presented.

CHAPTER 2

Service Oriented Architectures in Manufacturing Enterprises

Several passages in this chapter are reproduced verbatim from the following publications:

1. Ahmed Ismail and Wolfgang Kastner. Service-Oriented Architectures for Interoperability in Industrial Enterprises. In Stefan Biffl, Detlef Gerhard and Arndt Luder (eds.). Multi-Disciplinary Engineering for Cyber-Physical Production Systems. Springer International Publishing AG, May 2017.

2. Ahmed Ismail and Wolfgang Kastner. Surveying the Features of Industrial SOAs. In 2017 Annual Institute of Electrical and Electronics Engineers (IEEE) Industrial Electronics Society's 18th International Conference on Industrial Technology (ICIT). March 2017, 1-8.

3. Ahmed Ismail and Wolfgang Kastner. Coordinating Redundant OPC UA Servers. In 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). September 2017, 1-8.

Industrial enterprises are well-known for the heterogeneity of their technological landscapes. Development in such environments typically carries large engineering costs. These costs may be compounded by the technological choices and implementations of enterprise infrastructures, which commonly use monolithic applications to achieve their functional goals. The resulting system involves many implicit and explicit dependencies (e.g., to a technology stack) that introduce stiff resistance to change. In such a case, the enterprise cannot be described as agile or flexible. In fact, it is properties such as these and their implications that have become major arguments used by the proponents of SOAs to effect infrastructural changes in enterprises [33].

The SO paradigm attempts to reduce the overall costs by employing design features and patterns specifically geared towards the development of flexible, agile, and manageable systems of interoperable and reusable components. Consider for example the concept of a SO CPPS. CPPS are physical and computational resources that are tightly bound in coordinated and controlled relationships and embedded in a socio-technical context. The functionalities originally addressed by monolithic applications may be decomposed and distributed across the system's member devices. That is, by applying the

service-oriented design pattern, a large business problem can be fragmented into smaller problems that may then be solved using a number of small and related units of logic, termed services, rather than through a single monolithic application. Distributing these services across the system would create networks of smart devices that are inherently resilient due to their lack of dependence on any central component. Furthermore, as long as services are designed with standardised interfaces, system-wide interoperability is guaranteed. The internals of these services, such as how they are implemented, or what technology they use is hidden behind the service interface, thereby affording the system technological independence and flexibility. Services are also typically designed with functional agnosticism to allow for their reuse to reduce future application development efforts. Together, the concepts that define the SO approach, all of which are summarised in [28, 34, 35] and presented in Table 2.1, afford an enterprise the agility it requires and minimises the need for integration [36, 37, 34, 28].

According to [38], several projects have been formulated to research the application of SOA based solutions to manufacturing domain problems since 2003. However, the discourse in this chapter is limited to the more recent and completed projects within the time period of 2010 to 2017. The purpose of this chapter is to provide an overview of recent trends in the characteristics and technology choices of research-based SO RAs for the industrial domain. Although the chapter refrains from including research projects that are still ongoing at the time of this writing, an exception is made for the Arrowhead framework as it has 77 partners and a budget of 69 million Euros, making it one of the largest European research projects in the field of automation [38]. In total, five projects are surveyed, the Internet of Things at Work (IoT@Work), Production Logistics and Sustainability Cockpit (PLANTCockpit), ArchitecturE for Service-Oriented Process - Monitoring and Control (IMC-AESOP), Embedded systems Service-based Control for Open manufacturing and Process automation (eScop), and Arrowhead framework projects.

The analysis of the five projects is done from two different perspectives to determine the ease with which they may be translated into concrete architectures and the technologies that may be used in such realisations. The first is a more technical and detailed perspective that is achieved by extracting the specific protocols, standards, and specifications used to define the various parts of the architectures' communication stacks. The second perspective is addressed by applying a software RAs analysis framework developed in [32]. Together, a succinct overview provides a baseline understanding of the features of the respective architectures.

Characteristic	Sub-Characteristic	Explanation
Discoverability		Services are supported using metadata that allows them to be discovered and interpreted.
Modularity	Modular Decomposability	The equivalent concept of functional decomposition as applied to modules.
	Modular Composability	The ability to create a software service or system by freely combining reusable services.
	Modular Understandability	The function of a service should be comprehensible without requiring knowledge on any other services.
	Modular Continuity	A service interface should conceal service implementation details to allow changes to the service to occur without them requiring changes in other services.
	Modular Protection	Perimeterisation of modules to prevent the cascading of faults unto other services.
Interoperability		Ensured ability for different modules to communicate with each other.
Loose Coupling		Appropriately defined service contracts that increase the independence of services from their implementations and from each other.
Location Transparency		The decoupling of a service from a specific location allowing dynamic service lookups and runtime binding that enhance the system's flexibility, availability and performance.
Composability	Application	The piecing together of services and their orchestration using application logic to achieve specifically set goals.
	Service Federation	The aggregation of services under a single service representation.
	Service Contracts	The explicit definition of a service's features and parameters as contractual terms and conditions in a granular form accessible by service requesters. This may include the definition of supported data types, data models, policies and other features that declare a service's interaction requirements.
	Service Orchestration	Service execution as part of an application should be sub-transactional and not permitted to perform data commits. This allows the system to rollback to the pre-transactional state in case of service failure.

Table 2.1: Features of SOAs [28, 34, 35].

Name	Duration	Objective
IoT@Work	Jun 2010 - Jun 2013	Using IoT technologies to decouple application/control programming from the network, enabling communication-centric plug & work capabilities, and enhancing the system security [39]
PLANTCockpit	Sept 2010 - Dec 2013	Creating a SO and centralised plant-wide HMI [40].
IMC-AESOP	Sept 2010 - Dec 2013	Using SO approach for SCADA/DCS in large-scale process control systems [41].
eScop	Mar 2013 - Feb 2016	System integration using ontology based knowledge-management, embedded devices, and SOA [42].
Arrowhead	Mar 2013 - Feb 2017	Providing a SO technical framework for cooperative automation in technologically heterogeneous systems [43].

Table 2.2: A summary of project durations and objectives.

2.1 Five Preliminary SOAs

This section focuses on five service-oriented RAs that resulted from collaborations between research, vendor, and user organisations. To reiterate, these are the IoT@Work, eScop, PLANTCockpit, IMC-AESOP, and Arrowhead framework projects. The inferred or explicitly stated purpose of each of these projects is summarised in Table 2.2.

2.1.1 IoT@Work

The IoT@Work project represents its RA using layers and planes. In terms of the former, three layers are used; the physical, abstraction, and composite service layers, as shown in Fig. 2.1. The first of these, the physical layer, is the physical world and is therefore composed of physical devices. The second layer is an abstraction of the physical devices as resources and services. In the context of the IoT@Work architecture, a resource is an object representing a specific physical or virtual element. A service gives access to a resource by specifying the type, identifier and interface. Effectively, a single device may be represented using one or more resources and services. The third and final layer is that of composite services. These group together the elements of the second layer to hide their complexity and deal with context, contention over resources, and access rights. It is this third layer atop which applications such as event notification, Complex Event Processing (CEP), Network Access Control (NAC), and controller Input/Output (I/O) applications run [39].

To address the functional aspects of these three layers, the IoT@Work project defines a set of core services, listed and defined in Table 2.3, and organises a large number of functional components that compose them into three planes; the communication, security, and management planes. The communication plane is concerned with the orchestration of network resources and communication to resolve access contention issues and provide support for Quality of Service (QoS) guarantees. The security plane, as the name implies, manages and integrates security into the overall system. Lastly, the

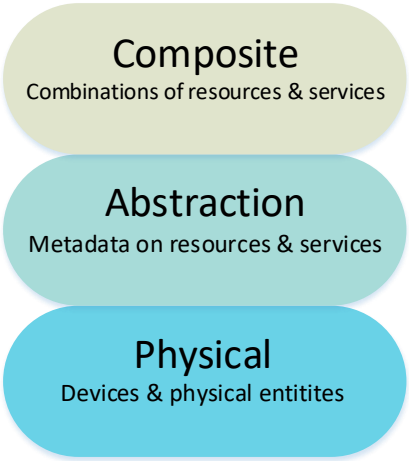

Figure 2.1: The three layers of an IoT@Work architecture [39].

management plane attends to device, service, and configuration management with a focus on their inter-relations [39].

2.1.2 PLANTCockpit

The PLANTCockpit system architecture, shown in Fig. 2.2, is composed of an internal and external system. The external system refers to the data sources connected to the PLANTCockpit using proprietary or open interfaces, such as an ERP system, OPC server, or sensors and actuators. As for the internal system, this consists of five layers: system connector, function engine, persistence, visualisation engine, and presentation engine [40].

The first of these, the system connector layer, is primarily concerned with interfacing with external data sources. It provides the configurable *adapter* modules required to allow the PLANTCockpit to access and communicate with these sources. Due to their configurability, an *adapter manager* is included in the architecture to oversee the entire life cycle of adapters. As for the external data structures acquired through the adapters, these are transformed to an internal data structure using a *mapper* module. Finally, the layer uses two generic components, the *subscriber* and *publisher*, to query the external systems via the adapters and push the data retrieved by way of the adapter and mapper components to the function engine layer, respectively [40].

The function engine layer, receiving these data, provides a platform atop where analytics and functions may be executed. It is based on the concept of *function blocks*, which, inspired by the object oriented paradigm and the IEC 61499 standard, are reconfigurable and encapsulated blocks of program code with clearly defined interfaces to allow for reuse and composability. These blocks' life cycles are managed using a *func-*

Core Service	Explanation
ENS	A common functional component that collects and distributes events.
ENS ARB	A broker between ENS clients attempting to access namespaces and the ENS AS.
ENS AS	Decision point for access requests sent to the ENS ARB.
PDP	Evaluates the status of capability tokens and policies to approve or refuse access requests.
Revocation Service	Manages capability revocation requests and capability revocation life cycles.
ENS Namespace Management Service	A service for the management of hierarchical structures used for the organisation of event publishing.
Slice Management System	A three part service consisting of a CSI, SEP and Slice Manager. Used for the creation of a 'slice'[1].
Embedded Application Configuration Service	Provides devices with the configurations required by their applications.
DS	Stores device information in an ontology-based DS data model.
Orchestrated Management	Orchestrated Management Authoring Support: A lightweight algorithm and API.
	Orchestrated Management Scheduling Service: Algorithms to produce management plans and schedule operations.
	Management Services: A wrapper around existing operations in the three planes so that they may be used and executed in Orchestrated Management scenarios.
	Context Services: Capture constraint values to provide context. May be a parameter of the MES or ERP system.
Complex Event Monitoring Service	Responsible for the verification of rule compliance to allow the system to meet safety and security goals.

[1]A slice is a virtual network with QoS guarantees and policies.

Table 2.3: The IoT@Work Core Services [39].

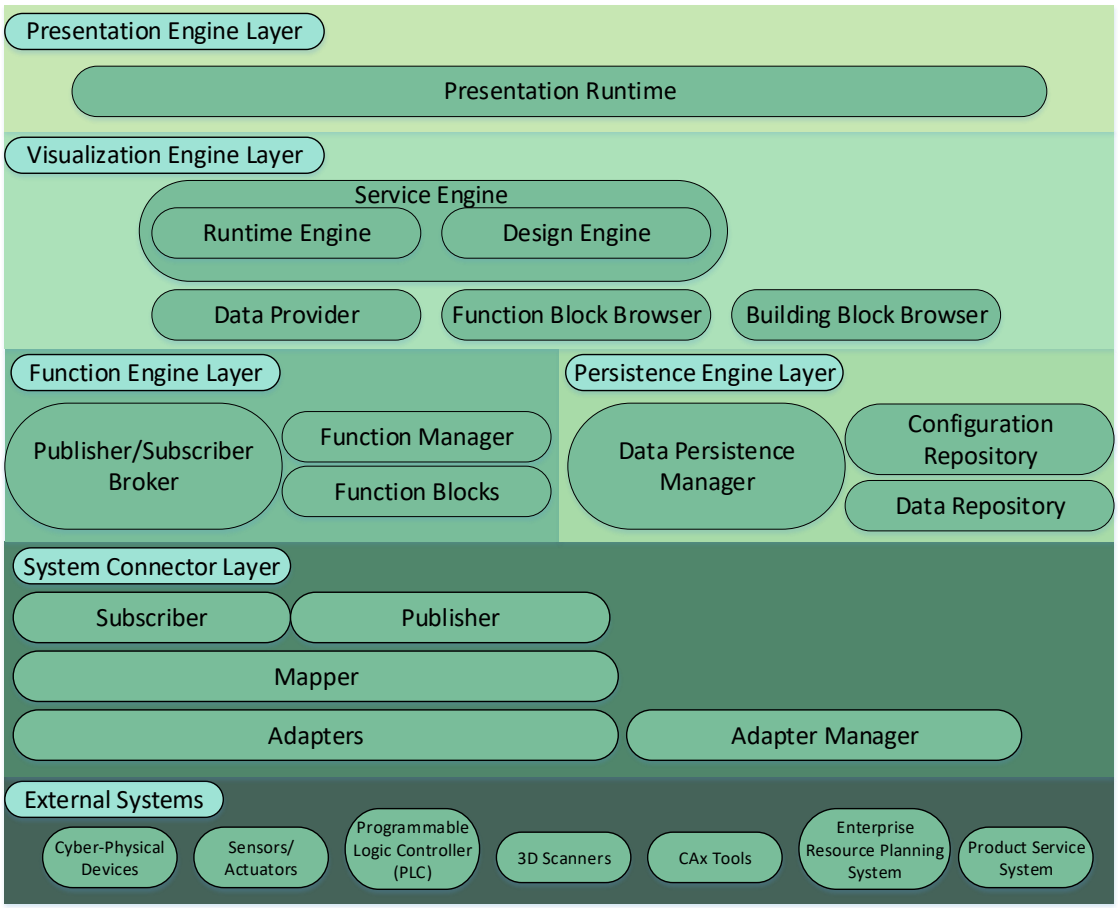

Figure 2.2: The PLANTCockpit system architecture [40].

tion manager, while a *pub/sub broker* (publisher/subscriber) provides them with secure, reliable, and event-driven mechanisms through which they may communicate with each other [40].

Any data relevant to the function engine or any other layer's workings are managed and stored using the persistence layer. This subsystem is composed of three components; the *data persistence manager*, *configuration repository*, and *data repository*. The manager administers the storage, archiving, retrieval, and deletion of data. The configuration repository maintains all of the data needed to configure the internal components of the PLANTCockpit system at design and runtime. Finally, the data repository stores all of the data required by analysis processes in the PLANTCockpit system. It includes a cache that can temporarily store data to improve the system performance, and a more permanent store that archives historical data [40].

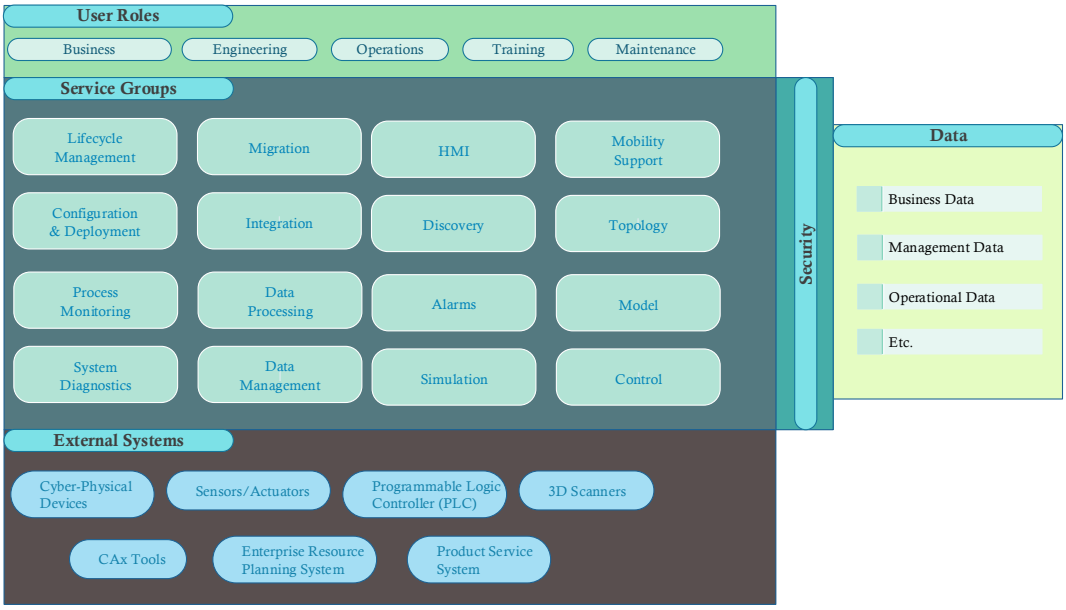

Figure 2.3: The IMC-AESOP system architecture [44].

Finally, any data to be presented via the HMI interface is prepared for visualisation using the visualisation engine layer. It consists of a service engine which is an aggregation of a runtime and design engine. The former contains the configurable User Interface (UI), while the latter configures the interface using a composition of *building blocks* (visualisation elements) and their associated data points. A *building block browser* and *function block browser* are used to make all possible building blocks and all available data points provided by the data persistence manager accessible by the design engine for the UI's configuration, respectively. Finally, a *data provider* component subscribes to the Pub/Sub Broker for data and events that it delivers to the runtime engines. The presentation engine layer, which is composed solely of a *presentation runtime engine*, then graphically presents the configured building blocks [40].

2.1.3 IMC-AESOP

As opposed to the PLANTCockpit framework, the IMC-AESOP architecture attempts to provide a generic architecture to support multiple applications, with the HMI only being one of these applications. As such, the framework is in fact a behemoth of services, service groups, and interactions presented using both natural language and semi-formal descriptions based on the Fundamental Modelling Concepts (FMC) graphical notation. The abridged description of the framework's components are shown in Table 2.4 [44].

Based on the architectural overview given in [44], the framework differentiates be-

tween four system components, as shown in Fig. 2.3. The first component is that of user roles, which designates user's *business*, *operations*, *engineering*, *maintenance*, or *training* roles. These roles interact with or impact the architecture directly or indirectly as they take part in plant processes. The second system component consists of the service groups. These act as the glue binding together the user roles, the external systems (the third component), and the plant data itself (the fourth) [44].

Service Group	Component Services	Explanation
Alarms	Alarm Configuration Alarm & Event Processing	Defines, maps, filters, and aggregates alarms based on the principles of CEP. Supports simple alarms. May be time and/or event/alarm-triggered.
Configuration & Deployment	System Configuration Service Configuration Service Configuration Repository	Responsible for the configuration, deployment, and enforcement of configurations on the various plant elements. The model repository service provides it with a structure for saving hierarchical configuration structures of nodes. The service may also manage the versioning of services and the instantiation of plant meta-models.
Control	Control Execution Engine	Typically a distributed service, it can execute process models and configurations and support the online reconfiguration of processes and hot-standby redundancy.
Data Management	Event Broker Data Consistency Actuator Output Historian Sensory Data Acquisition	Responsible for "data retrieval, consistency checking, storage, searching", basic eventing, and actuator output control. The service connects data producers with higher level services and provides methods for mapping data to the appropriate data models and ontologies.
Data Processing	Filtering Calculation Engine CEP Service	Provides services for basic filtering, CEP and calculations. The CEP engine allows for low-latency analysis of event data. It provides a management interface that allows for the creation, update or removal of rules used for processing events. The calculation engine supports users in executing numerical or logical operations over process data.
Discovery	Discovery Service Service Registry	Supports the discovery of system components by type and location. Uses a registry to support the discovery of services implemented using technologies without inherent discovery capabilities, and to allow for discovery by remote entities where multicast and broadcast based discovery would be of limited usefulness.
HMI	Graphics Presentation	Provides a generic web interface where graphical tools may be embedded.
Integration	Composition Service Service Mediator Gateway Business Process Management and Execution Service Model Mapping Service	Responsible for ensuring interoperability between heterogeneous components using translation and encapsulation. Also serves as a platform for the execution of business processes as service compositions and their presentation as higher level services.
Life cycle Management	Code Repository Life cycle Management Service	Covers "aspects such as maintenance policies, versioning, service management and also concepts around staging (e.g. test, validation, simulation, production)". It also contains a code repository service which maintains the source code of services to allow

		for service maintenance, deployment, upgrade, and other functions.
Migration	Infrastructure Migration Solver Migration Execution Service	Responsible for the migration of legacy systems to the SOA-based approach by identifying system dependencies, offering migration strategies, and executing them.
Mobility Support	Mobile Service Management	Concerned with the management of mobile assets and so is responsible for asset tracking, address mapping, data synchronisation, and similar supporting functions.
Model	Model Management Service Model Repository Service	Consists of generic services for the management of models, a repository to structure these models, and an interface to the repository.
Process Monitoring	Monitoring Service	Provides HMIs with an entry point into the system. Responsible for gathering information from the physical process using other system components and semantically enriching it, and for handling alarms and events.
Security	Security Management Service Policy Management Service	Enforces and executes security measures for confidentiality, authentication, and other features. Also defines and manages security rules/policies for identity or role-based access control and for the definition of identity federation.
Simulation	Constraint Evaluation Simulation Scenario Manager Simulation Execution Process Simulation Service	Is connected to almost every other service group as it simulates systems and their processes, evaluating constraints and simulating execution. It consumes other systems' exposed simulation endpoints to imitate characteristic system performance and behaviour features.
System Diagnostics	Asset Diagnostics Management Asset Monitor	Concerned with monitoring the status and health of plant assets and mainly used for maintenance and planning.
Topology	Naming Service Network Management Service Location Service	Allows for reporting and management on the system's physical and logical features. It includes Domain Name System (DNS) functionality, device discovery and management, and asset location services.

Table 2.4: IMC-AESOP Service Groups & Services [44].

2.1.4 eScop

The eScop project is composed of five layers, a *Physical Layer (PHL)*, *Representation Layer (RPL)*, *Orchestration Layer (ORL)*, *Visualisation Layer (VIS)*, and *Interface Layer (INT)*. The PHL is concerned with the physical equipment in the eScop system and therefore provides device and service descriptions. The information provided by the PHL is consumed by the RPL, which is responsible for knowledge representation. Syntactic and semantic service descriptions based on the service implementations are mapped and stored in the RPL. The ORL, which coordinates and executes service compositions, in addition to requiring service descriptions from the RPL, may also need input from the PHL to successfully orchestrate the execution of services. The VIS, configured by the RPL, then provides an interface for the user to interact with the system data that

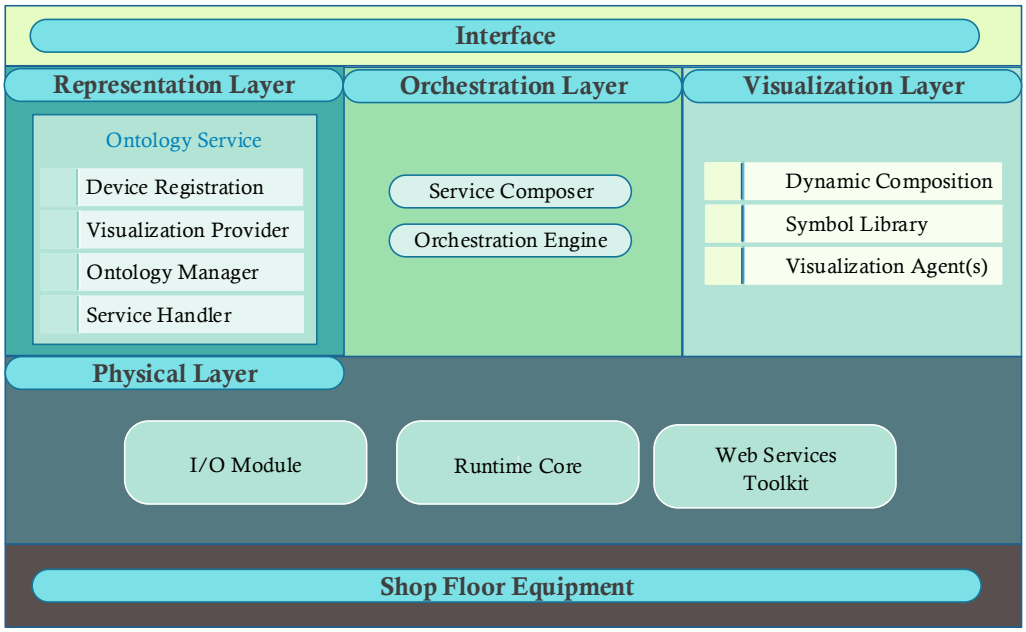

Figure 2.4: The eScop system architecture [45].

is accessible via the PHL. Finally, the INT acts as the entry point for external systems and services in the eScop architecture and is functionally concerned with the provision of technology adapters and access control measures [45].

As for the features of the RA, the system's services define concrete technologies for implementation. For example, the various components of each of the layers are described in the next few paragraphs.

Starting from the bottom-up, the PHL consists of an *I/O module*, *runtime core*, and *Web Services (WS) toolkit*. These support connections to the physical devices, the definition of applications on controllers, and provide the devices with web services and notification mechanisms, respectively [45].

The RPL achieves its goal of knowledge representation using an *ontology service*, a set of functions, and the ontology itself. The ontology service consists of four modules: *device registration*, *visualisation provider*, *ontology manager*, and *service handler* modules. These handle device registration and de-registration in the ontology, assist with visualisation, provide an interface for the configuration or editing of the model, or manage the RPL's connections to the various components of other layers. The governance of access to the ontology in the triplestore is done by the RPL's *SPARQL Protocol* and *Query Language (SPARQL) query factory* and *ontology connector* internal functions, and it is suggested that, once secured, SPARQL-over-HTTP may be used for these factors. As for the ontology itself, it is stipulated as being the eScop Manufacturing Systems Ontology

(MSO), which is a proprietary component created by one of the designing members of the architecture [45, 46].

The ORL coordinates the various components in the architecture using a *service composer* and an *orchestration engine*. The former maps process definitions to configurations applicable to the system, and the latter executes them [46].

The VIS aims to allow for flexible and generic graphical interfaces. For this, it needs a *dynamic composition* module, a *symbol library*, and *visualisation agent(s)*. Together, the VIS is able to map descriptions from the RPL to visualisation elements from the symbol library which are then transformed by the agent(s) into web pages that can be displayed using a web browser [45, 46].

2.1.5 Arrowhead Framework

The Arrowhead project divides its framework into three parts that are *design guidelines*, *documentation guidelines*, and a *software framework*. The first provides a description of design patterns for making legacy or newly created application systems compliant with the Arrowhead Framework. The documentation guidelines provide templates for the description of services, systems, and system-of-systems. The software framework is the main concern of this section [43].

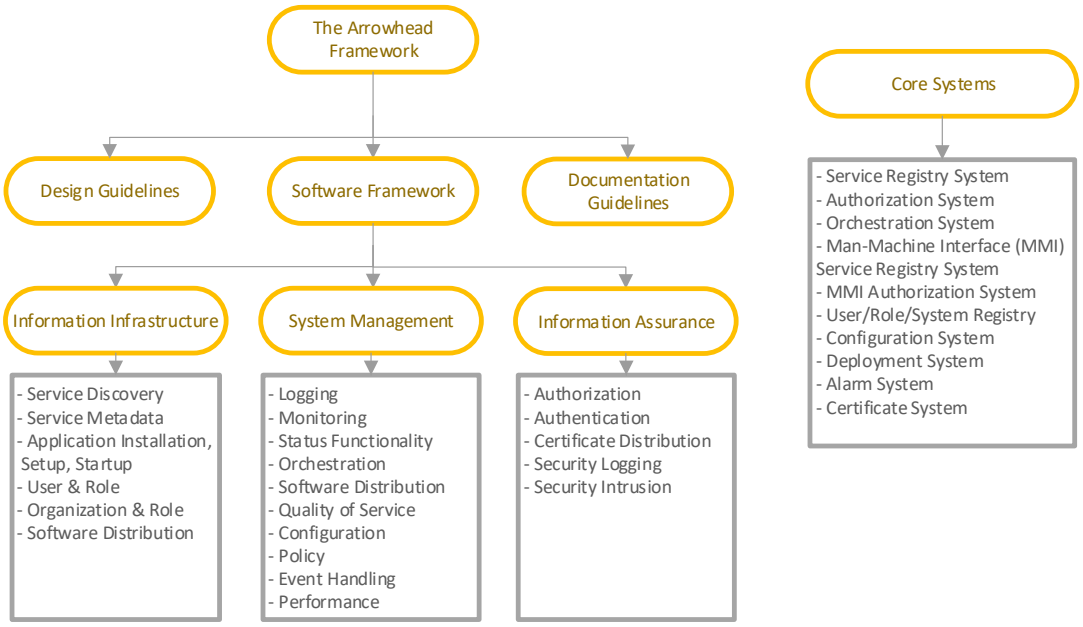

Figure 2.5: The Arrowhead framework [43, 47].

The software architecture defines a grouping of core services. These services are meant to support communication exchanges between domain-specific application services. These core services and systems are effectively divided into three groups: *Information Infrastructure (II)*, *Systems Management (SM)*, and *Information Assurance (IA)*. II provides service descriptions and information on how to connect to services and systems. SM core services are concerned with orchestration and system-of-systems composition. Finally, IA addresses security and safety factors in information exchange processes. The categorisation of core services and systems identified by the framework under these three groups is shown in Fig. 2.5 [43, 48].

2.2 Realisations of the Reference Architectures

So far, the RAs have been described in an abstract manner. This portion of the chapter inspects the technology stacks implemented by each of the architectures. For comparability, these technologies are segregated into categories that address the various functional aspects addressed by all of the architectures. Brief descriptions are given on both the mature and novel technologies implemented for each category to give a succinct overview of the technical properties of each project in the pursuit of achieving interoperability. A summary of the technology stacks of the five projects is shown in Table 2.5. For improved accessibility, the row of technologies employed by each layer is reproduced at the start of each subsection.

2.2.1 Service Discovery

An essential aspect present in any service-oriented architecture is service discovery. Due to the close association of the Device Profile for Web Services (DPWS) with SOAs, the use of the Web Services Dynamic Discovery (WS-Discovery) specification is common. Four out of the five architectures, excepting eScop, either directly implemented or discussed methods to allow for the use of the WS-Discovery protocol.

The WS-Discovery protocol is based on the use of multicast messages (typically Simple Object Access Protocol (SOAP) over User Datagram Protocol (UDP)) to announce or probe for services using specially crafted eXtensible Markup Language (XML) documents. Announcements operate using multicast *Hello* and best-effort *Bye* messages. Likewise, *Probe* and *Resolve* messages are also multicast. The former is used to locate services based on service types and/or scopes. The latter searches for a specific service by name. The use of a *Discovery Proxy* is encouraged to allow for the active suppression of multicast traffic in the network. The specification also endorses the caching of multicast service advertisements to incur further savings. Finally, with respect to securing the discovery process, the specification does not require, but recommends the use of unique XML signatures and a number of other properties to mitigate against a variety of attacks [49].

		IoT@Work	PLANTCockpit	IMC-AESOP	eScop	Arrowhead
Service discovery		WS-Discovery, UA Discovery, Representational State Transfer (REST) Directory Service	WS-Discovery	WS-Discovery, UA Discovery, CoAP	Custom	DNS, DNS-SD, mDNS, XML-over-HTTP, JSON-over-HTTP
Service description		WS-Discovery, WS-MetadataExchange, WSDL, WS-Transfer, OPC UA	WSDL	WSDL	Swagger	DNS-SD, WADL
Data representation & access		Custom ontology (uCode), OPC UA, SNMP Management Information Base (MIB)	Custom database schema, XML schema	OPC UA, SenML	Manufacturing Systems Ontology	OPC UA, SenML, Home Performance XML, CoRE Link Format, ThingML
Information encoding		XML, JSON, OPC UA Binary, HTML, SAML, XACML	XML, OPC UA Binary(?), HTML	XML, JSON, EXI, XOP/MTOM, OPC UA Binary	XML, JSON, YAML	XML, JSON, EXI
Message exchange		DPWS, HTTP, AMQP, OPC UA Binary profile	JMS	HTTP/HTTPS, CoAP, OPC UA Binary profile, UA web services profile	-	HTTP/HTTPS, CoAP, XMPP, MQTT, OPC UA
Networking, data link, and media	Application Services	NTP, IEEE 1588 PTP	-	NTP, IEEE 1588 PTP, IEEE 802.1Q, SNMP, DHCP, DNS, LLDP, STP	-	NTP
	Networking	IPv4, IPv6	-	IPv4, IPv6	-	IPv4, IPv6
	Media & Data Link	NFC, QR, Profinet	-	6LoWPAN, IEEE 802.15.4, IEEE 802.11, RS-485 Modbus, Profibus	-	6LoWPAN, IEEE 802.15.4, IEEE 802.11p, NFC, UWB, GSM, GPRS, UMTS, RS-485 CAN
Security		CBAC SAML, XACML, Digital Signature & XML encryption specifications, IEEE 802.1AR, IEEE 802.1X	JMS interceptor, LDAP SSO	HTTP basic authentication and RBAC	Input sanitisation, testing required	DNS TSIG, DNSSEC, X.509 certs., TLS/DTLS, PKI, 'ESTADO' system, MPI + SSH

Table 2.5: The technology stacks of the 5 reviewed architectures.

Service Discovery				
IoT@Work	PLANTCockpit	IMC-AESOP	eScop	Arrowhead
WS-Discovery, UA Discovery, RESTful Directory Service	WS-Discovery	WS-Discovery, UA Discovery, CoAP	Custom	DNS, DNS-SD, mDNS, XML-over-HTTP, JSON-over-HTTP

Table 2.6: The service discovery technologies employed in the 5 reviewed architectures.

The IMC-AESOP is one of the architectures implementing the WS-Discovery protocol directly, for example, to allow Service Bus instances to discover each other [50]. However, one of the main contributions of IMC-AESOP hinges on its bridging of DPWS with the industry-focused OPC UA standard. OPC UA is a set of specifications that mainly define an information model and a set of services to interact with it (further information on OPC UA can be found in Section 2.4). This includes a discovery service. The architecture therefore presents concepts for supplementing OPC UA's discovery mechanisms using WS-Discovery. To elaborate, the OPC UA discovery protocol requires the use of a discovery server. The address of the server must be known beforehand by participating OPC UA clients and servers. The IMC-AESOP approach presents two methods for auto-discovery in OPC UA systems using WS-Discovery. The first involves the use of WS-Discovery to allow OPC UA clients and servers to automatically find the OPC UA Discovery Server. The second approach involves replacing the OPC UA Discovery Server with the WS-Discovery protocol to allow OPC UA clients to find OPC UA servers directly [51].

A second core technology in IMC-AESOP is the *Constrained Application Protocol (CoAP)*. Identified as a suitable protocol for device-level integration of constrained devices, such as those belonging to Wireless Sensor Networks (WSN), the IMC-AESOP approach discusses a reliance on the discovery mechanisms of CoAP, CoAP multicast, and the Constrained RESTful Environments (CoRE) Resource Directory (RD), for the location of services and resources hosted on resource-limited clients [52, 53].

Since the PLANTCockpit architecture is dependent on adapters to interface with the various systems and function blocks, the discovery mechanism employed is dependent on the system being interfaced. For example, the system implements a DPWS adapter to allow for the discovery of DPWS devices. The adapters themselves, however, are implemented as function blocks based on the concepts of IEC 61499 Function Blocks (FB). The identification of FBs depends on FB Service Interfaces, and these are implemented using the OSGi framework. As such, although not explicitly stated, it may be the case that the implementation depends on OSGi's service registry to register, get or listen for services [54, 55].

Although the IoT@Work approach explored and compared the WS-Discovery and OPC-UA's discovery mechanisms, it did so within the context of auto-configuration. Instead, the IoT@Work system uses a configurable RESTful DS as a form of service registry. Devices interact with the DS using a RESTful API to retrieve, submit or delete

Service Description				
IoT@Work	PLANTCockpit	IMC-AESOP	eScop	Arrowhead
WS-Discovery, WS-MetadataExchange, WSDL, WS-Transfer, OPC UA	WSDL	WSDL	Swagger	DNS-SD, WADL

Table 2.7: The service description technologies employed in the 5 reviewed architectures.

device and service information using Hypertext Transfer Protocol (HTTP) GET, PUT, POST, and DELETE requests. The RESTful nature of the service allows every service to be modelled as a Uniform Resource Locator (URL) accessible resource. The system also supports the use of Quick Response (QR) codes and Near-Field Communication (NFC) tags for the identification of devices [39].

Similar to IoT@Work, the eScop project generates its own discovery mechanisms. Discovery here is based on the multicasting of Hello, Bye, and Probe messages. In this respect, it is similar to the WS-Discovery specification [56]. Inspection of the source code[1], however, shows that the protocol does not follow the WS-Discovery specification. This is because a few critical differences exist, such as the use of JavaScript Object Notation (JSON) encoding for messages, multicast IPs, and ports different than those stipulated for use by WS-Discovery.

Finally, the Arrowhead project defines three approaches for service discovery. The first is a service registry functionality based on the DNS and Domain Name System Service Discovery (DNS-SD). Effectively, DNS is a hierarchical database mechanism that can store any kind of data and DNS-SD is a method for specifying how DNS resource records may be named, structured, and browsed. These records may be accessed using unicast DNS requests or Multicast Domain Name System (mDNS). The Arrowhead framework applies mDNS for constrained devices, such as those belonging to WSNs. The second and third approaches for discovery in the Arrowhead projects are based on the use of XML-over-HTTP and JSON-over-HTTP for RESTful web services. The former uses a DNS protocol specific to Arrowhead to allow for service discovery and the retrieval of service and data descriptions. The JSON-over-HTTP approach is marked as future work and an implementation still remains to be published. The framework does however discuss the prospect of implementing a translation service for integration between the XML/JSON and DNS-SD registry systems [57, 58, 59, 60, 61, 62, 63].

2.2.2 Service Description

This subsection focuses specifically on the aspect of service contracts as defined in Table 2.1. The goal of service contracts is to define a minimal level of interoperability and thereby reduce the need for integration. It may do so by making available definitions of the service's functionality, data model, data transfer mechanisms and encodings, and

[1] http://www.escop-project.eu/tools/

policies for security and quality of service, amongst other things. Naturally, these contracts themselves need to be interpretable by all services available in the registry or operating environment. Similar to what is the case for service discovery, a web services technology, the Web Services Description Language (WSDL), is employed by a number of projects [64].

The WSDL language is a machine-readable XML-based language for the definition of interfaces. It is capable of describing all of a service's operations, the data required and output by each operation, and their respective data types. It may also provide addressing and networking information to support inter-service connectivity. Both of the IMC-AESOP and PLANTCockpit projects use WSDL files for service descriptions. However, for describing sensor services, the IMC-AESOP project uses the JSON, XML and Efficient XML Interchange (EXI) compatible Sensor Markup Language (SenML) [65, 44].

The eScop project also employs a WS-based approach and develops a WS-enabled Remote Terminal Unit (RTU) titled the eScopRTU. Within the eScopRTU, all services are IEC 61131 Structured Text Language (STL) functions that are used to execute operations on resources. They are RESTful, hypermedia-driven, accessible via a REST API, and the API documentation (read "service descriptions") are created using Swagger. Swagger is a specification for the definition of language-agnostic, human and machine-readable representations of RESTful APIs. The specification requires that the API be described using either JSON or YAML Ain't Markup Language (YAML)[2]. The resulting files may then be processed by tools that can generate clients in a variety of languages. The Swagger ecosystem also includes tools to display and test the API. Swagger has since been renamed the OpenAPI Specification (OAS) [66, 67].

IoT@Work, as previously mentioned, explored the prospects of auto-configuration using WS-Discovery and OPC UA. Part of the procedure outlined involves the acquisition of service descriptions. With WS-Discovery, this was achieved by having metadata on a DPWS-enabled device retrieved by its controller using the WS-Transfer protocol. The metadata that may be included is defined as part of the WS-MetadataExchange specification. This metadata would allow the service to share WSDL definitions, XML schema, policy expressions, and so on. For the case of OPC UA, the GetEndpoints Service, which is part of the OPC UA standard, retrieves the information required to allow for secure communication between clients and servers. The information mainly consists of addressing and security policies and definitions [68, 69, 70].

Service description in Arrowhead is dependent on the method of service discovery that is implemented. As previously mentioned, the DNS system uses DNS-SD guidelines to organise the resource records. In such a case, the specification already allots a structure for the definition of addressing, service name, and other connection-related information. For any additional requirements, the DNS TXT record is capable of accommodating such information. The XML-based system uses XML schema for the description of data, and the Web Application Description Language (WADL), which is WSDL's counterpart

[2]http://www.yaml.org/

Data Representation and Access				
IoT@Work	PLANTCockpit	IMC-AESOP	eScop	Arrowhead
Custom ontology (uCode), OPC UA, SNMP MIB	Custom database schema, XML schema	OPC UA, SenML	Manufacturing Systems Ontology	OPC UA, SenML, Home Performance XML, CoRE Link Format, ThingML

Table 2.8: The data representation and data access technologies employed in the 5 reviewed architectures.

for RESTful services for the description of service interfaces. The JSON-based system, as previously mentioned, is yet to be defined [60, 61].

2.2.3 Data Representation and Access

Data representation and access have been previously discussed with realisations of service descriptions in Section 2.2.2. Here, the focus is a more in-depth description of the information or data model and the semantics used by the architectures' respective implementations. The main purpose of these models and semantics is to homogenise the representation of information or data in the entire system ensuring accessibility by all participants and avoid the need for any data transformation [64].

Of the five projects only two, IMC-AESOP and Arrowhead, rely primarily on mature standardised models for the representation of data. In IMC-AESOP, the system applies the OPC UA information model to link up information in the majority of the enterprise, except for the lowest layer, which instead uses a custom data model based on the SenML. The former, OPC UA, contains a flexible address space that can be used to create information models that capture objects, their attributes, and relationships. These objects are known as nodes in the OPC UA address space and can be used to represent physical or virtual components. The resulting information models create full-mesh networks of these nodes, with associated properties and relations, and are exposed to applications through OPC UA servers [44, 70, 71].

As for SenML, the associated specification defines a data model suitable for highly constrained devices, such as sensors and actuators. It does so by having a minimalist approach where the goal is to maximise the amount of information not included in a message while still allowing for self-describing data that includes measurements and metadata. The result is a single array data model that contains a series of data records. The records can contain the device's unique identifier, a time stamp, the measurement value and unit, amongst other details. This allows for the description of the measurements and device, in addition to the measurement values themselves. The IMC-AESOP project, however, claims that the information granularity level that can be carried by SenML's data model is insufficient for its needs and therefore creates a custom data model that is primarily based on SenML to achieve this granularity [72, 44].

Similar to IMC-AESOP, Arrowhead also identifies OPC UA and SenML as suitable candidates for the implementation of data structures and semantics. However, it also

highlights the Home Performance XML standard and the CoRE Link Format as appropriate for its needs. The former is a set of data standards that define a number of XML schema and associated data elements to allow for the description of customers, contractors, buildings (and their components and systems), and energy performance factors such as conservation, consumption, and savings, both as actual readings and as estimates. The goal of these standards is therefore, as the name implies, standardisation in the collection and transfer of information in the domain of home performance. The latter, the CoRE Link Format, is a realisation for exposing the Uniform Resource Identifier (URI) of resources on constrained devices and networks. It does so by extending the HTTP Link Header format to include the URI descriptions, such as resource relations and attributes, as a message payload, and specifying an entry point URI as a default request path for the retrieval of these URIs. However, the existence of divergences from the semantic and modelling technologies listed in the Arrowhead guidelines as the energy production demonstrator pilot adopts the domain-specific language, Thing Markup Language (ThingML), instead of the ones listed above for its semantic needs [47, 73, 74, 75, 76].

In contrast to IMC-AESOP and Arrowhead, eScop and IoT@Work are the ontology-driven models. eScop, as previously mentioned, develops a proprietary MSO based on OWL to describe the system components, their attributes, and relationships. The MSO is the evolved form of the Politecnico di Milano Production Systems Ontology (P-PSO), which is a general taxonomy for discrete manufacturing systems. The MSO extends the P-PSO to include logistics and process production from the perspective of control. The MSO also incorporates concepts to allow for the visualisation of the respective systems and their data. The information is stored in a Resource Description Framework (RDF) triple-store database that supports SPARQL-over-HTTP to allow for web-based interactions with the ontology [77, 78, 79].

The IoT@Work project also follows an ontology based approach by storing information on devices in a DS-specific data model (ontology) that uses an RDF-triple-store. In addition to RDF, the model is also inspired by the uCode Relation Model that models device profile attributes as subject-predicate-object triples. Effectively, the resulting DS model is a connected directed graph where the vertices are physical or virtual entities or primitive elements and the edges in the graph represent the relationships between the various entities and elements. The DS is also capable of validating and handling requests for information on devices acquired through an exposed RESTful interface. This information may be retrieved from the database or by collecting it directly from connected devices. The IoT@Work-compliant devices, unlike the DS, use the OPC UA address space and information model. The project also supports the retrieval of information from Simple Network Management Protocol (SNMP) compliant devices. Mappings between the respective device and DS models are therefore a necessity [39, 80].

The PLANTCockpit project presents its own metamodel for a database schema and an XML schema for the storage of visualisation engine configurations. The database schema consists of four customisable data types and runtime data types to allow for the persistence of analytics-relevant data. The XML schema contains several elements

Information and Message Encoding				
IoT@Work	PLANTCockpit	IMC-AESOP	eScop	Arrowhead
XML, JSON, OPC UA Binary, HTML, SAML, XACML	XML, OPC UA Binary(?), HTML	XML, JSON, EXI, XOP/MTOM, OPC UA Binary	XML, JSON, YAML	XML, JSON, EXI

Table 2.9: The information and message encoding technologies employed in the 5 reviewed architectures.

to allow for the rendering of SVG components in HTML5 pages, and their linking to data points to create a configurable and compound HMI made up of different graphical elements [81].

2.2.4 Information & Message Encoding

For message encoding, all of the projects employ XML and extend support for one or more other specifications. XML is a platform-independent data structuring format that defines rules for the textual encoding of human and machine-readable data. It allows for user-defined tags and different data types and processing methods. By allowing for the definition of syntax rules and standardised contracts through the use of Document Type Definition (DTD) or XML Schema Definition (XSD) descriptions, the validation and verification of encoded data structures are possible. Several specifications have since been defined for the binary-encoding of XML documents to address the overhead and performance issues associated with XML. Of the possible choices, the EXI specification is employed by the IMC-AESOP and Arrowhead projects for the compact exchange of information [82, 83, 47, 44, 39, 84].

Other than XML, JSON is also widely employed, being used by all but PLANTCockpit. Stipulations in the PLANTCockpit approach do, however, allow for the inclusion of JSON. This is the case, for example, with the configuration connector module which is required to be format-agnostic in handling configuration data. As for the JSON specification itself, JSON, like XML, is a data structuring specification that defines rules for the formatting of exchangeable and human and machine-readable data. Tools for the parsing and generation of JSON exist for a large number of programming languages therefore making it a popular alternative to XML. It follows a minimalist encoding approach, using a small number of characters to denote the structure and value of data. Similar to XML, JSON allows for the definition of JSON-based schema for the validation of resulting encodings. Binary encodings, such as the Concise Binary Object Representation (CBOR) specification, exist for JSON. However, aside from the mention of CBOR support as a long-term goal for Arrowhead's historian, it does not appear as though any of the projects include a binary encoding for JSON in their respective stacks [44, 85, 47, 86, 87, 88, 89, 90].

Aside from XML, EXI, and JSON, three other formats used are OPC UA Binary, HTML, and XML-binary Optimized Packaging and Message Transmission Optimisation Mechanism (XOP/MTOM). OPC UA Binary, as the name implies, is the binary protocol

for OPC UA. Like other binary representations, it is the performance and overhead-sensitive data format for OPC UA. It is used in both IoT@Work and IMC-AESOP and is considered to be part of the PLANTCockpit OPC UA adapter. HTML, on the other hand, is used for the structuring and presentation of multimedia web content using human and machine-readable semantic descriptions. It is explicitly stated as part of the visualisation layers of IoT@Work and PLANTCockpit. Finally, XOP/MTOM, is used by IMC-AESOP for the transmission and reception of binary data in SOAP messages [39, 91, 81, 44, 92, 65].

Further specifications include security relevant ones, such as the XML-based Security Assertion Markup Language (SAML) and eXtensible Access Control Markup Language (XACML), which are employed in IoT@Work, SOAP for the structuring of messages, and the previously mentioned YAML for the description of RESTful APIs. The first of these, SAML, is a standard for the communication of data for authentication and authorisation, while XACML handles the definition of access policies. How these are applied as part of IoT@Work will be discussed later in Section 2.2.7 in this chapter. SOAP defines a platform independent XML-based framework for message structuring, encoding, and processing and for the representation of RPC and responses. The SOAP message structure consists of a SOAP envelope, SOAP body, and, optionally a SOAP header. The first, the SOAP envelope, is used to represent the message itself, while the SOAP header can be used to add features and their associated attributes to a message. Lastly, the SOAP body contains the message contents to be conveyed to the other communicating parties. The platform-agnostic nature of the SOAP protocol is one of the driving factors behind its popularity in the web services community. As for YAML, this was discussed earlier as the language of choice for the configuration of Swagger files under the eScop project. YAML, like its counterparts, is a serialisation language aimed at minimizing the number of characters required to indicate the structure and value of data while maximizing human readability in the resulting data interchange format. It is built around a typing system, an aliasing mechanism and primitives such as mappings, scalars and sequences. According to the YAML specification, compared to JSON, it is more difficult to generate and parse but more legible to humans than JSON. The specification also states that there is no direct correlation between XML and YAML [39, 93, 94, 82, 95, 96].

2.2.5 Message Exchange

For message exchange, web-based solutions (such as HTTP and CoAP) and Message Oriented Middleware (MOM) (such as Java Message Service (JMS), Advanced Message Queuing Protocol (AMQP), Extensible Messaging and Presence Protocol (XMPP), and Message Queue Telemetry Transport (MQTT)) are preferred.

Starting with HTTP, this application-level, stateless, and generic protocol is used by all projects for the transfer of hypermedia across networks. The protocol typically runs over the Transmission Control Protocol (TCP), employs MIME-like messages for communication, and uses URIs to provide access to resources. For its secure equivalent, HTTP over SSL/TLS (HTTPS), HTTP is transported over a Transport Layer Security

Message Exchange				
IoT@Work	PLANTCockpit	IMC-AESOP	eScop	Arrowhead
DPWS, HTTP, AMQP, OPC UA Binary profile	JMS	HTTP/HTTPS, CoAP, OPC UA Binary profile, UA web services profile	-	HTTP/HTTPS, CoAP, XMPP, MQTT, OPC UA

Table 2.10: The message exchange technologies employed in the 5 reviewed architectures.

(TLS) tunnel. In several instances, such as IMC-AESOP and PLANTCockpit, implementations used HTTP/HTTPS for the conveyance of SOAP messages. CoAP, on the other hand, is a low overhead URI-based web protocol for M2M communication over UDP with support for unicast, multicast, proxying, caching, stateless HTTP mapping, and binding to Datagram Transport Layer Security (DTLS). Due to properties such as these, a number of projects, namely IMC-AESOP and Arrowhead, have favoured the use of the CoAP protocol for the access of constrained devices and network implementations in their realisations [97, 98, 47, 85, 44, 65, 99, 100, 90, 101].

As for the MOMs employed, PLANTCockpit uses JMS, IoT@Work employs AMQP, and certain Arrowhead demonstrators implement XMPP or MQTT. MOMs are a paradigm for asynchronous, loosely coupled and reliable communication in distributed systems. These properties, as well as others such as high scalability and availability, are enabled through the use of an intermediate layer, the middleware, that handles the messaging process on behalf of the communicating parties. The first MOM to be discussed is JMS, which is a vendor-agnostic standard that defines a Java API and semantics for the description of the interface and the messaging system behaviour. It therefore allows applications to communicate with heterogeneous enterprise messaging systems, and simplifies their development process. However, the implementation of the messaging service itself is not defined by the standard. As such, only a very general structure for the JMS message is defined by the standard necessitating the inclusion of integration techniques if multiple implementations of MOM exist within the same system [102, 103, 82, 91, 80, 104, 90, 101].

In contrast to JMS, AMQP defines an open-standard messaging protocol that includes the networking protocol and message structure and remains agnostic towards the client API and message broker employed. Its protocol provides flow control features, message delivery guarantees, and highly flexible routing mechanisms for communicating parties. It appears that IoT@Work based its decision of using AMQP for its ENS implementation on the fact that it addresses, as a standard, both aspects of high-level modelling and wire-level communication concepts [105, 106, 107, 80].

In terms of Arrowhead's choices, the XMPP and MQTT protocols are either highlighted for use or directly implemented in a number of its services and demonstrator pilots. The former, XMPP is a widely-implemented XML and TCP/IP based protocol for near-Real Time (RT) communication. The protocol addresses aspects related to connection establishment and tear-down, security, discovery, reliability, messaging, and

inter-entity interactions. MQTT, on the other hand, is a lightweight protocol for constrained and unreliable systems that is more efficient than HTTPS, but is not extensible, and does not natively include connection security, transactions, discovery, or message fragmentation features. Multiple instances in the Arrowhead documents list the desire for services to support both MQTT and XMPP, or, in the case of the mediator service, to support translation between the two protocols [108, 104, 90, 101, 109, 110].

Other protocols noted include DPWS and OPC UA. DPWS uses SOAP-over-HTTP and SOAP-over-UDP bindings, yet certain member services, such as WS-Discovery and WS-Transfer, are transport independent. With SOAP-over-HTTP the SOAP message is placed inside the HTTP payload field for request/response messaging allowing for features from both specifications. Similarly, the SOAP-over-UDP binding allows for the inheritance of UDP's messaging, encoding, security and other mechanisms.

Regarding OPC UA, four combinations of encoding, security, and transport protocols are possible:

- UA Binary + UA-SecureConversation + UA-TCP
- UA Binary + HTTPS
- UA XML + SOAP + HTTPS
- UA XML + WS-SecureConversation + SOAP + HTTP

The first of these compositions, referred to as native UA Binary, is mandatory for implementation. Both IMC-AESOP and IoT@Work use the native UA Binary profile for the transport layer of their OPC UA implementations. IMC-AESOP, however, also includes a communication interface in its DPWS stack that implements the third profile that consists of UA XML, SOAP and HTTPS, labelling it as OPC UA over WS. Arrowhead presents a proof of concept of a condition monitoring system that employs the OPC UA software development kit (SDK) from Unified Automation, and discusses its use for the integration of legacy components in its framework. Finally, although OPC UA is highlighted as a development technology for PLANTCockpit, the details of the implementation are unclear from the public deliverables [111, 112, 113, 44, 70, 51, 47, 108, 114, 65].

2.2.6 Networking, Data Link and Media

As may have been partially visible from the previous sections of this chapter, a wide and heterogeneous variety of media and their associated specifications are used, required, or discussed for possible implementations. Effectively, however, only three projects, Arrowhead, IMC-AESOP, and IoT@Work, explicitly state the technologies implemented at the networking, data link, and media layers.

The Arrowhead framework's stack is declared as being comprised of Internet Protocol version 4 (IPv4), Internet Protocol version 6 (IPv6), IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN), 802.11p, 802.15.4, NFC, Ultra-Wide Bandwidth (UWB), and Network Time Protocol (NTP). However, other than the aforementioned protocols, several more are used in one of the pilot demonstrations. Specifically, the

Networking, Data Link and Media			
Layer	IMC-AESOP	IoT@Work	Arrowhead Framework
Application Services	NTP, IEEE 1588 PTP	NTP, IEEE 1588 PTP, IEEE 802.1Q, SNMP, DHCP, DNS-SD, LLDP, STP	NTP
Networking	IPv4, IPv6	IPv4, IPv6	IPv4, IPv6
Data Link & Media	6LoWPAN, IEEE 802.15.4, IEEE 802.11, RS-485 Modbus, Profibus	NFC, QR, Profinet	6LoWPAN, IEEE 802.15.4, IEEE 802.11p, NFC, UWB, GSM, GPRS, UMTS, RS-485 CAN

Table 2.11: The Networking, Media & Data Link standards and supporting application services [47, 115, 116, 117, 118, 44, 39, 119, 80].

Global System for Mobile Communications (GSM) (2G), General Packet Radio Service (GPRS) (2.5G), Universal Mobile Telecommunications System (UMTS) (3G), WiFi, and RS-485 Controller Area Network (CAN) standards are highlighted by the Arrowhead project as possible solutions for communication in electrical vehicle charging infrastructure. The wireless specifications allow users to communicate with charging stations using devices separate from the vehicle, such as mobile devices. So far, the pilot demonstration has limited its implementation to UMTS and WiFi. As for the communication channel between the vehicle and the charging station, the Arrowhead project uses CAN, basing it on the CHAdeMO standard [47, 115, 116, 117, 118].

The IMC-AESOP project declares a stack somewhat similar to Arrowhead, using IPv4, IPv6, TCP, UDP, 6LoWPAN, IEEE 802.15.4, IEEE 802.11, and NTP. However, it uses RS-485 Modbus instead of CAN, and also includes Profibus, UA Native, and IEEE 1588 Precision Time Protocol (PTP). The majority of these protocols, namely IPv4, IPv6, TCP, UDP, 6LoWPAN are part of the DPWSCore stack in IMC-AESOP. The component responsible for bridging the DPWS and OPC UA stacks implements the UA Native protocol. A pilot demonstrating the migration of a plant's lubrication system to the IMC-AESOP approach used the Modbus protocol to connect to the distributed control system and a specific stack consisting of XML/EXI, CoAP, NTP, UDP, IP, 6LoWPAN, IEEE 802.15.4. A second pilot, for building system of systems with SOA technology highlights the integration of smart home systems with communication infrastructure using SenML, EXI, CoAP, IPv6, IEEE 802.11, IEEE 802.15.4 and cellphone communication technologies (agnostic) [44].

The IoT@Work approach necessitates the use of IPv6 and designates IPv4 as optional. For its DS, as previously discussed, both NFC and QR codes are supported. For timing, both NTP or IEEE 1588 PTP are possible. To demonstrate the auto-configuration system, it uses the RT Ethernet standard Profinet. Its network slices technology is mapped using Ethernet Virtual Local Area Network (VLAN). IoT@Work differs from other approaches with its focus on protocols such as SNMP, Dynamic Host Configura-

Security				
IoT@Work	PLANTCockpit	IMC-AESOP	eScop	Arrowhead
CBAC, SAML, XACML, Digital Signature & XML encryption specifications, IEEE 802.1AR, IEEE 802.1X	JMS interceptor, LDAP SSO	HTTP basic authentication and rbac	Input sanitisation, testing required	DNS TSIG, DNSSEC, X.509 certs., TLS/DTLS, PKI, 'ESTADO' system, MPI + SSH

Table 2.12: The security technologies employed in the 5 reviewed architectures.

tion Protocol (DHCP), DNS, LLDP, and Spanning Tree Protocol (STP) to support the network-level autoconfiguration of devices [39, 119, 80].

2.2.7 Security

The IoT@Work approach to security is founded on Capability-Based Access Control (CBAC). This mechanism centres around the use of transmissible tokens that reference an element and its access rights. A process in possession of a valid token may therefore interact with the referenced element within the constraints of its access rights. These capabilities may be forged or revoked in the form of XML documents following specific schema with elements borrowed from the SAML, XACML, Digital Signature and XML encryption schema. IoT@Work also requires that devices have secure identifiers based on the IEEE 802.1AR specification, and that authentication for NAC be carried out based on IEEE 802.1X. With these mechanisms, the system designs and employs multiple components in a SO fashion to perform NAC and to secure the system's event namespace [120, 39, 121].

The PLANTCockpit project explored the possibility of using single-sign on solutions for a security service. Ultimately, the PLANTCockpit approach employs a Lightweight Directory Access Protocol (LDAP) Single Sign-On (SSO) service for access control and the management of user rights. Interactions with the LDAP service are through a Java client implemented using the JLDAP library. User access rights govern the components and data sources that a user is permitted to interact with. As for the security aspects related to PLANTCockpit's use of JMS, notes in the deliverables claim that PLANTCockpit permits the encryption of the JMS message body, with the encryption to be managed using a central component and that JMS security is addressed using 'interceptors'. Unfortunately, the security aspects of PLANTCockpit are addressed in a non-public deliverable (D3.2) and, as such, details on the interceptors and other system mechanisms for security are not available for further analysis [86, 122, 82].

Published materials show that Arrowhead instils security measures for its Service Discovery and Authorisation Control, for mediation in legacy systems, and for communication in general. The Service Discovery service is secured by using Domain Name System Transaction SIGnature (DNS TSIG) keys for DNS updates and Domain Name System Security Extensions (DNSSEC) for queries. Authorisation Control is depen-

dent on the use of X.509 certificates, and provides TLS security for the establishment of secure communication links. In the case of Arrowhead's Virtual Market of Energy pilot, the Authorisation module uses Public Key Infrastructure (PKI) and X.509 certificates over REST for authentication and XMPP-over-TLS for encrypted communications. Generally, the consensus throughout the Arrowhead framework is to secure information exchange using TLS. In the case of UDP communication, DTLS is highlighted as the applicable counter-part. Another project goal is to develop a secure NFC interface for industrial applications. Based on a publication, this goal may have been addressed through the 'ESTADO' system for smart maintenance. This system uses a 'CUT-IN' module that consists of a secure and a non-secure, but more powerful, controller. The module is designed to be an add-on that would provide security features to 'non-smart' and legacy devices. This would include abilities for the secure storage and execution of data, integrity checks, encrypted memory, and encrypted in-CPU calculations. Finally, Message Passing Interface (MPI) with Secure Shell (SSH) for protected communications are used in the implementation of a distributed framework for 3D swarming systems such as aerial vehicles and WSNs [60, 123, 47, 124, 125, 126, 127].

In the case of IMC-AESOP, by self-admission, "cyber-security was not at the heart of [this] project" [44]. As such, IMC-AESOP states that WS-Security and WS-Reliable Messaging were not implemented as part of its DPWS communication stack, and neither was IPsec included in the IPv6-based stack for WSANs, claiming them all as planned additions. Furthermore, the service bus in IMC-AESOP only implemented HTTP basic authentication and Role-Based Access Control (RBAC) for service calls with such rights only being given to administrative users. In accordance with RFC 7617, unless communication takes place within a secure system (e.g., over TLS), basic authentication is not to be considered secure as credentials are transferred in clear-text. It is unclear if IMC-AESOP implements basic authentication over a secure channel [44, 128].

Similarly, other than security testing and a high level discussion of defining and applying a security model as part of an SOA ecosystem, eScop did not address the aspect of security. In the case of the former, security testing of the RPL is defined as a method for verifying the robustness of the ontology by employing "ad-hoc offensive queries". For the VIS layer, testing is to be applied to access control measures. As for the testing of the integration of the PHL, RPL, VIS and ORL, this entailed ensuring that data in the system is secured and that the functionality of the system cannot be misused [129, 130].

2.3 Comparison & Discussion

The concept of SOAs has been in existence for well over a decade and is generally considered to be a stable and tried architectural design pattern. This chapter has so far inspected the architectural designs and technological choices of five preliminary SO RAs. For a definitive measure of comparison, the final examination takes place using an analysis framework developed in [32]. This framework consists of two components. The first defines a classification method for categorising architectures based on context,

goals, and design. These factors are addressed using a set of interrogatives for which there are a select number of possible values, some of which are mutually exclusive. The factors, interrogatives, and architectural attributes are summarised in Table 2.13 [32].

The second element of the framework is a set of five architectural templates and two sub-types that define preset compositions of values from the elements of Table 2.13. An architecture that shares the same attributes as a category belongs to its type. The type most relevant to the forthcoming analysis is Type 5. This is because the RAs are facilitation architectures that use preliminary technologies and are developed through collaborations between research centres and industrial partners. The combination of values for the type 5 category and the degree to which the five RAs match are shown in Table 2.14.

As may be noted from Table 2.14, all of the architectures reviewed are facilitation RAs, include preliminary technologies, and are designed by partnerships between research, industrial software design, and user organisations for application in multiple organisations. In all of the remaining sub-dimensions, however, nearly all of the RAs have divergent properties.

For the D1 dimension, certain RAs are either over or under specified. The IoT@Work, eScop, and Arrowhead fit the former description, while IMC-AESOP and PLANTCockpit are of the latter type.

The D2 dimension is notably one of the more difficult dimensions for classification. The framework's authors note that the classification technique applied is subjective and therefore inherently imprecise. However, the deviations noted for IMC-AESOP and PLANTCockpit are irrefutable as certain elements needed by the D2 dimension are not defined by the respective RAs. The remainder of the RAs all have detailed specifications for their components, and all but eScop detail their algorithms and protocols.

For the D3 dimension, IMC-AESOP defines its architecture in a completely abstract manner, while the remaining four all specify concrete elements in their architectures [40].

In terms of representation, IMC-AESOP, eScop, and Arrowhead all use semi-formal techniques. The first uses the FMC graphical notation, the second the Unified Modelling Language (UML), and the last the Systems Modelling Language (SysML) [44, 129]. As for IoT@Work and PLANTCockpit, neither architecture explicitly declares its use of any specification for representation.

The importance of this analysis is born of previous results in [32], which note that a lack of congruence between a RA and its category makes the RA vulnerable to low adoption rates and criticisms by stakeholders. Due to the lack of reporting on adoption rates, it is more prudent to refrain from making the same conclusion. However, the authors of [32] also note that the presence of ambiguities in RAs, such as the informal representation of components, leads to a need for additional documents to clarify the architecture. The technology stacks detailed in this chapter, and extracted from numerous publications, deliverables, and other materials, may assist in achieving such clarity. In certain cases, where the implementations did not address or make available certain layers of their technology stacks, extra effort may still be necessary in the carrying out of concrete implementations.

Factor	Dimension	Value	Explanation
Goal	G1: Why [1]	Standardisation	Interoperability-focused concrete architectures.
		Facilitation	Guidelines for the design of concrete architectures.
Context	C1: Where [1]	Single Organisation	The architecture is developed to standardise or facilitate the development of software for a single enterprise.
		Multiple Organisations	The architecture is to be used by several organisations that have a common property (industrial domain, technological constraints etc.)
	C2: Who	Software organisations	Apply the RA; requirements providers that may also be involved in the design of the RA.
		User organisations	Users of the resulting software; requirements providers.
		Independent organisations	Refers to research, standardisation, non-governmental and other organisations that do not implement or use the resulting software solutions.
	C3: When [1]	Preliminary	The technologies required for the application of the RA exist only as research experiments, proof-of-concepts, or are partially or completely absent at the time that the architecture is defined.
		Classical	The technologies required for the application of the RA are mature at the time that the architecture is designed.
Design	D1: What	(See explanation)	The element types defined. Possible options are "components and connectors, interfaces, protocols, algorithms, and policies and guidelines"[32]
	How	D2: Detail	Detailed: Many elements belonging to three or more aggregation levels
			Aggregated: A single aggregation level containing a small number of elements.
			Semi-detailed: Between detailed and aggregated.
		D3: Concreteness	Abstract: General definitions of architectural elements.
			Semi-concrete: Specific definition of a selection of options for every component of the architecture.
			Concrete: Defines the option to be used from the selection for every element in the architecture.
		D4: Representation	Informal: The architecture is defined using ambiguous graphical notation and/or through natural language.
			Semi-formal: Uses a non-ambiguous graphical notation that is not based on a mathematical foundation.
			Formal: Uses clearly-defined formal specifications based on mathematical techniques.

[1]Possible values for these dimensions are mutually exclusive.

Table 2.13: A summary of the applied analysis framework [32].

Category	Type 5	IoT@Work	PLANTCockpit	IMC-AESOP	eScop	Arrowhead
G1:	Facilitation	X	X	X	X	X
C1:	Multiple organisations	X	X	X	X	X
C2:	Research Centres (D) Software design organisations (D, R) User Organisations (R)	X	X	X	X	X
C3:	Preliminary	X	X	X	X	X
D1:	Components, algorithms, protocols	≈	≈	≈	≈	≈
D2:	Detailed/Semi-detailed: components, algorithms Aggregated/semi-detailed: protocols	≈	≈	≈	≈	≈
D3:	Abstract	-	-	X	-	-
D4:	Formal/Semi-formal	-	-	X	X	X

Notation: X means it is a match, ≈ means deviations exist, and - means it does not match [32].

Table 2.14: Attributes of a type 5 RA [32] and degree of match of the analysed RAs.

To counter these short-comings, this dissertation can focus on delivering an additional SOA, albeit, with a higher degree of congruence with the findings of [32]. However, rather than contribute to the proliferation of SOAs by creating an additional competing architecture, it is the opinion of this author that the community would be better served by selecting and improving upon a SO communication architecture that already achieved standardisation, currently enjoys wide adoption by industry, and is well-supported by academic research as well. Thus, the remainder of this dissertation will focus on the enhancement of manufacturing infrastructure using SO middleware based on the OPC UA specifications family. The next section therefore presents an overview of OPC UA and highlights its characteristic properties.

2.4 OPC UA

In its simplest terms, the OPC UA standard is composed of a set of specifications for the definition of data transfer software interfaces in a client-server architecture. An example of the overall structure of OPC UA is shown in Fig. 2.6. The remainder of this section will give an overview of OPC UA's technology stack, a summary of which is available in Table 2.15.

	OPC UA
Service discovery	LDS, LDS-ME, GDS, ServerCapabilityIdentifiers
Service description	Standard Services, Profiles, information model traversal
Data representation & access	OPC UA information model
Information encoding	OPC UA Binary and OPC UA XML
Message exchange, networking, data link, and media	Native OPC UA Binary, OPC UA Webservices, and hybrid models based on OPC UA TCP, SOAP/HTTP, and HTTPS.
Security	Message SecurityProtocols with message signing and encryption. X509 version 3 in X690 DER format conformant with RFC 3280. Kerberos Authentication. Message sequence IDs. UserIdentityTokens with user/password combinations, X509v3 certificates, or WS-SecurityTokens. Event auditing.

Table 2.15: An overview of OPC UA's technology stack.

2.4.1 Service Discovery

Service discovery is defined in Part 12 of the OPC UA specifications [131] and is primarily concerned with two factors:

1. finding servers on the network
2. learning how to connect to them.

Mechanisms that make servers discoverable and that allow clients to discover them are therefore required.

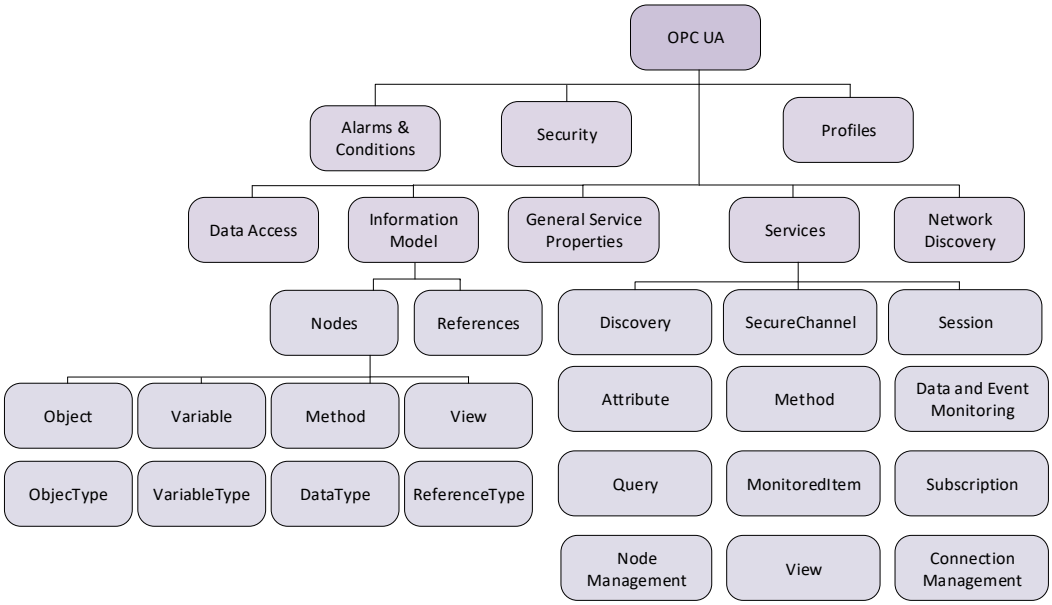

Figure 2.6: An interpretation of the overall structure of OPC UA.

Starting with the former, servers can be discovered after they register with a designated discovery server. There are three types of discovery servers possible: a LocalDiscoveryServer, LocalDiscoveryServer with MulticastExtension (LDS-ME), and a GlobalDiscoveryServer (GDS). It's worth noting that OPC UA allows servers to opt out of using in-band discovery, permitting them to instead announce themselves using out-of-band methods and services.

All three discovery servers keep identifying information on the OPC UA servers that register with them. An LDS is usually used by servers that are located on the same host. It follows that a LDS-ME registers servers that announce themselves in the same multicast subnet. A GDS registers OPC UA applications in a given administrative domain. It is possible for an LDS to register with a GDS. The GDS then periodically polls the LDS to update its database.

The possible options for server discovery available to OPC UA clients are:

1. using an out-of-band service,

2. executing a service call to find the servers on an LDS,

3. executing a multicast query natively from the client or via an LDS-ME, and

4. searching on the GDS.

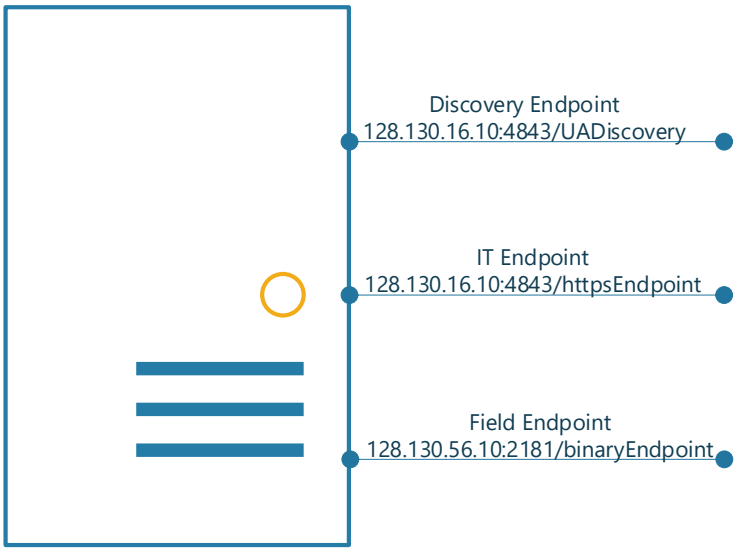

Figure 2.7: An example device with discovery, OPC UA HTTPS, and OPC UA TCP endpoints.

The mechanisms used for out-of-band discovery are service-specific and will not be discussed further. The discovery process for the latter three, however, operate as follows.

A server registers itself periodically on the discovery server. The information given to the discovery server may include the server and product URI, name, type, and the discovery endpoint. Of these, the discovery endpoint is especially important. An endpoint is an entry path to a device. OPC UA devices typically have at least two endpoints: one discovery endpoint with a well-known address path and one endpoint that supports a specific server communication profile. Given the discovery endpoint of a device, a client may enumerate the other endpoints on that device and connect to the one with a matching communication profile.

For example, referring to Fig. 2.7, a co-located Server is shown with three endpoints. The topmost demonstrates the discovery endpoint while the latter two provides alternate profiles for communication with clients using different transport, security, and encoding mechanisms. Thus, a client, may issue a service call (FindServers2 or FindServers) to a discovery server to identify the servers registered on it. Once the client locates the server it wants to connect to, it issues a second service call (GetEndpoints) to the discovery endpoint to request the endpoints on that device. The client then selects an endpoint with a compatible communication profile and connects to the device.

It is worth noting that, in searching for a server, the client may also include identifiers which describe certain features that may be offered by registered servers. This allows clients to filter for servers that can offer current data, historical data, alarming features,

Service Set	Description
Discovery	A set of services implemented by OPC UA servers and dedicated discovery servers. These allow clients to find servers and determine the configurations needed to connect to them.
SecureChannel	Services that allow a client to establish a secure communication channel with a server with guarantees for message confidentiality and integrity. These are included in the OPC UA application's communication stack.
Session	Services that can be used by a client to establish an application-layer connection with a server.
NodeManagement	These services are used to interact with the AddressSpace. A client may use them to add, modify, or delete Nodes that form part of the information model on an OPC UA server.
View	Allows clients to browse the Nodes in a View. A View is a publicly defined portion of the AddressSpace hosted on a server.
Query	The Query service can be used to access the AddressSpace of a server without any knowledge on the server's logical schema. It allows Clients to filter a View for a select subset of Nodes.
Attribute	This service set allows Clients to read and write to the OPC UA defined "primitive characteristics of Nodes", also known as Attributes [132].
Method	Provides the means by which exposed function calls on an OPC UA server may be executed by clients.
Subscription	This allows clients to subscribe to monitored items on an OPC UA server and receive notifications on their values or status.
MonitoredItem	Used by clients to define items that can then be subscribed to for data and events.

Table 2.16: An overview of OPC UA's different service sets [132].

or specific information models before connecting to them.

2.4.2 Service Description

Service description in OPC UA can be seen as a collection of different aspects from the specifications working together. First of all, OPC UA provides a fixed set of services that can be used to establish communications between servers and clients and allow clients to interact with the application and information model. These services are divided by function into different service groups, which are referred to as service sets in OPC UA. The defined service sets are Discovery, SecureChannel, Session, NodeManagement, View, Query, Attribute, Method, Subscription, and MonitoredItem. A description of each service group can be found in Table 2.16. The service groups and their constituent services and mechanisms are standardised in the specification documents. Thus, there is no ambiguity related to their functionality and methods of operation [133].

Secondly, OPC UA defines a number of Profiles that group together different features that an OPC UA client or server may choose to support. A Profile is composed of one or more ConformanceUnits. A ConformanceUnit comprises a group of Services and/or information models. This allows vendors to develop compliant and testable OPC UA

applications. Profiles may be used to specify the functions of an OPC UA client and servers, as well as transport and security related functions. Hence, determining the profile of an OPC UA client or server allows an application to determine the majority of aspects typically found in a service contract. This may be done in the second part of the discovery process when a client calls the GetEndpoints service [134].

Finally, ad hoc services can be implemented as functions and exposed as methods on an OPC UA server. Therefore, they form part of the server's address space and can be accessed and manipulated using standard OPC UA services. Information specific to the function of the method, such as the service description, inputs, and outputs, may be inferred from the information model, e.g., by traversing the address space.

2.4.3 Data Representation & Access

The OPC UA information model provides a flexible address space for modelling, exposing, and consuming networks of data and metadata of varying degrees of complexity. To do so, the OPC UA standard bases its model on two elementary units, *Nodes* and *References*.

Nodes are the simplest units of information and can be of various *NodeClasses* (types) that are predefined by the standard with specific *Attributes* (properties). The available NodeClasses are "*Object*, *ObjectType*, *Variable*, *VariableType*, *DataType*, *ReferenceType*, *Method*, and *View*" [135]. An Object node represents an abstract or physical component in the modelled system. A Variable node is used to store a value. A Method node is used to expose a function so that it may be called remotely. A View node is used to specify a subset of an address space. Last of all, the ObjectType, VariableType, DataType, and ReferenceType nodes, as their names imply, are used for the specification of Object, Variable, data, and Reference types, respectively.

Every node in the address space is identified and addressed using a unique *NodeId*. In addition to the canonical NodeId, a node may also have alternative NodeIds. Each NodeId is composed of an identifier and a *Namespace*. The namespace is used to allow naming authorities, such as vendors and organisations, to define unique NodeIds.

References, on the other hand, are used to connect nodes for organisational or filtration purposes. Each reference therefore has a source and destination node, a direction, and a *ReferenceType* used to indicate the properties and meaning of a reference.

2.4.4 Information & Message Encoding

OPC UA defines two data encodings: OPC UA Binary and OPC UA XML. These encodings are defined for 25 built-in data types. OPC UA Binary, as was stated in Subsection 2.2.4, is the performance and overhead-sensitive format. OPC UA XML is the WS compatibility protocol and uses the XML Schema Part 2 [136] to define the formats or syntax for encoding the built-in types.

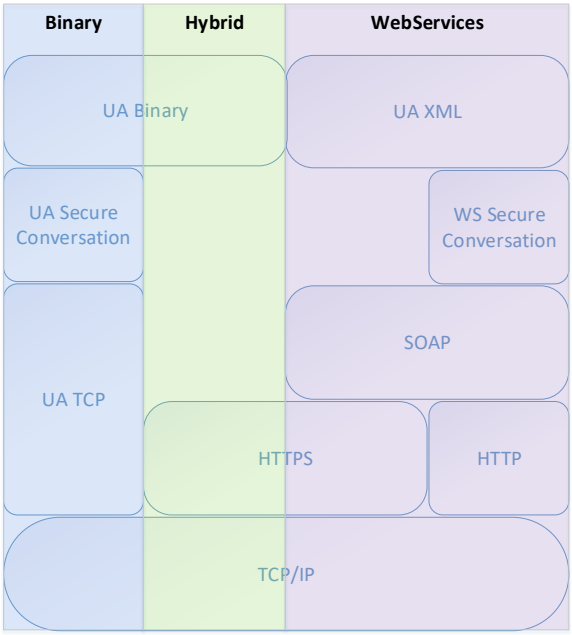

Figure 2.8: The possible protocol bindings for OPC UA.

2.4.5 Message Exchange, Networking, Data Link and Media

For message exchange, OPC UA provides three approaches. These are the Native OPC UA Binary, OPC UA Webservices, and hybrid communication models. These protocol bindings address the encoding, security, and transport layers, as shown in Fig. 2.8. The three transport protocols used are OPC UA TCP, SOAP/HTTP, and HTTPS. OPC UA is agnostic to the data link and media layer and only singles out NTP for clock synchronisation in [137].

OPC UA TCP is a connection-oriented protocol based on TCP that is compatible with the OPC UA SecureChannel services for secure communications. The protocol defines the message headers, message structure, and mechanisms necessary to establish and close a connection, as well as error handling and error recovery. [137].

The SOAP/HTTP protocol is deprecated in OPC UA Version 1.03 of the standard due to a lack of adoption by industry [137].

The HTTPS protocol has previously been introduced in Subsection 2.2.5. However, OPC UA introduces a custom header termed OPCUA-SecurityPolicy to inform the OPC UA server of the SecurityPolicy in use by the client. OPC UA uses the protocol to carry XML or OPC UA Binary encoded messages over a secure channel.

2.4.6 Security

Security is an integral part of OPC UA as evidenced by Part 2 of the specification being dedicated solely to the definition of security objectives, threats, and integrated solutions for the standard. This subsection will give a brief overview on some of the security features of OPC UA.

OPC UA integrates Message SecurityProtocols for application authentication, message confidentiality, and message integrity. These protocols depend on the SecureChannel service group to define the services needed to open and close secure channels for communication. Three SecurityModes are supported by the protocols: None, Sign, and SignAndEncrypt. The SecurityProtocols use X509 version 3 certificates encoded in X690 Distinguished Encoding Rules (DER) format. The X509 certificates used by OPC UA applications should conform to the RFC 3280 profile for Internet applications. Security policies are also defined to specify the algorithms used during a handshake. OPC UA also explicitly supports the use of the Kerberos Authentication Service [137].

WS SecureConversation, similar to the SOAP/HTTP protocol, is deprecated in OPC UA Version 1.03 due to a lack of adoption by industry [137].

OPC UA SecureConversation, which is defined as the "binary version of WS-SecureConversation, is still supported by the standard for secure non-SOAP and non-XML communication. OPC UA SecureConversation segments messages into separate chunks that have a 4-byte sequence assigned to detect and prevent replay attacks [137].

While OPC UA application authentication is handled using X509v3 certificates, user authentication uses UserIdentityTokens that may be user/password combinations, X509v3 certificates, or WS-SecurityTokens. OPC UA also gives some support for user authorisation, e.g., in the form of error codes that can signify a problem in the authorisation process. OPC UA does not, however, specify mechanisms for user authorisation [138].

Finally, OPC UA also defines the events to be logged when the different service groups are used by applications in support of auditing functions [133].

2.5 Classification & Discussion

This section is concerned with classifying the OPC UA architecture according to the Angelov *et al.* [32] analysis framework to determine its attributes and vulnerabilities.

First off, OPC UA is a standardisation architecture for application in multiple organisations and is primarily concerned with system interoperability. By definition, this designates it as a Type 1 architecture in the Angelov *et al.* [32] framework. The degree of match of OPC UA with the attributes defined by [32] for a Type 1 RA is shown in Table 2.17.

Starting with the C2 dimension, the design of the OPC UA architecture is led by the Open Platform Communications (OPC) Foundation, which is a standardisation organisation composed of a consortium of software, user, and independent organisations. It therefore gives stakeholders the ability to contribute to the specification process and

Category	Type 1	OPC UA
G1:	Standardisation	X
C1:	Multiple organisations	X
C2:	Standardisation Organisation (D) Software organisations (R) User Organisations (R)	X
C3:	Classical	X
D1:	Components, interfaces, policies/guidelines	X
D2:	Detailed/Semi-detailed: interfaces Aggregated: components, policies/guidelines	X
D3:	Abstract	X
D4:	Semi-formal	X

Notation: X means it is a match, ≈ means deviations exist, and - means it does not match [32].

Table 2.17: Attributes of a type 1 RA and degree of match of the OPC UA specifications [32].

to moderate conflicting views. One example of each type of organisation, as extracted from the OPC Foundation's members list[3] includes ProSys, Inc.[4], Pfizer Inc.[5], and TU Vienna[6], respectively. While research or academic organisations, such as TU Vienna, are not shown as part of the C2 dimension, the Angelov *et al.* framework explicitly permits contributions by research organisations in the form of reviews and surveys for Type 1 architectures [32].

In reference to the C3 dimension, Table 2.15 demonstrates that the technologies adopted by OPC UA for implementation are mature solutions, thereby implying a classical nature for the OPC UA RA.

For the design dimensions D1 and D2, OPC UA defines detailed interfaces and aggregated components, policies, and guidelines for the implementation of a compliant system. While it may be debated that OPC UA also defines protocols to "demonstrate the interactions among the components" [32], the defined protocols are considered elements of the detailed interfaces that are incorporated to guarantee precise interoperability and the conformance of interactions between components in resulting implementations. Thus, the OPC UA architecture is also conformant in both the D1 and D2 dimensions.

In terms of concreteness, the OPC UA architecture only defines its components from a high level. No "monopolistic" approach that implies preference to a specific software organisation is applied in the specifications. OPC UA can therefore be stated as being abstract in the D3 dimension.

Finally, the OPC UA specifications demonstrate the use of UML, where applicable. Hence, OPC UA conforms to the Type 1 requirement for semi-formal notation in the

[3]https://opcfoundation.org/members
[4]https://www.prosys.com/
[5]http://www.pfizer.com
[6]http://www.tuwien.ac.at

D4 dimension.

To conclude, OPC UA is compliant with all of the identified dimensions for a Type 1 architecture. By extrapolating the results of Angelov *et al.*'s previous results in [32], as was done in Section 2.3, it appears that OPC UA therefore lacks the same vulnerabilities to criticisms and low adoption rates demonstrated by the previously reviewed five SO RAs. This is evidenced further by the fact that it is also well-accepted by industry and academia. OPC UA therefore appears as the appropriate candidate for the realisation of the SO envisioned system. Therefore, OPC UA will serve as the base technology of choice for the remainder of this dissertation. The coming chapters will proceed in a bottom-up approach by first demonstrating an alternate transport layer for OPC UA in Chapter 3 and then developing specific enhancements for OPC UA deployments in Chapter 4.

CHAPTER 3

Overlay Networking in Manufacturing Enterprises

Several passages in this chapter are reproduced verbatim from the following publications:

1. Ahmed Ismail and Wolfgang Kastner. Co-operative peer-to-peer systems for industrial middleware. In 2016 IEEE World Conference on Factory Communication Systems (WFCS). May 2016, 1-8.

2. Ahmed Ismail and Wolfgang Kastner. Discovery in SOA-Governed Industrial Middleware with mDNS and DNS-SD. In 2016 21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). September 2016, 1-8.

3. Ahmed Ismail and Wolfgang Kastner. Vertical Integration in Industrial Enterprises and Distributed Middleware. International Journal of Internet Protocol Technology 9(2/3):7989, 2016.

Thematically, the governing concept for the envisioned system can be considered under the umbrella term of "design for failure". This is an architectural paradigm that expects systems to be survivable by design. This means that they should be capable of sustaining business operations in the presence of failure by continuing to serve essential services. This chapter proceeds in congruence with this concept by developing a communication technology that is resilient to failures for manufacturing infrastructure.

The communication mechanisms must be designed with special consideration to the existing infrastructure of industrial enterprises. This is because, as previously mentioned in Chapter 1, industrial networking infrastructure is typically engineered based on numerous standards and binding legal constraints, found in [139] and summarised in Table 3.1, that may not be violated for the sake of connectivity. Generally, these constraints translate to a network design that follows a hierarchical and layered architecture, as was shown in Fig. 1.1, with strict controls applied to communication flows between said layers, while intra-layer communication is permitted to flow freely [22]. Superimposed upon the transport protocol, this means that the protocol should be able to mirror and maintain fidelity to such an architecture, while also having the flexibility to autonomously

Factor	Explanation
Regulations & Legislation	Systems must observe the imposed local, regional, national or international regulations, legislation and standards.
Impacts	The impact of system malfunctions on the business must reflect on the system design.
Safety	The system must comply with safety best practices. Standards such as IEC 61508 and IEC 61511, provide the necessary guidance.
Security	The system must be designed in accordance with the requirements of security risk assessments.
Locality	The physical location of systems must be taken into consideration when defining both the zones and the digital system in itself.
Architecture	The system must integrate within the overall technical and landscape architecture.
Operations & Maintenance	Operational and maintenance considerations impacting the system, the technical architecture, or the physical infrastructure, should be taken into account.
Organisation	The organisational structure and culture introduces a set of requirements that ought to be reflected unto the system under design.

Table 3.1: Governing factors in zonal population [139].

adapt to changing system requirements. The resulting network would therefore effectively be an 'overlay network', which is a network that constructs itself upon existing physical infrastructure [140].

Of the various subclasses of overlay networking that exist, one of the most well-developed and popular ones is P2P networks. Briefly, in P2P networks devices are connected together and share resources using direct exchange in a manner that is resilient against failures and transient population sizes, all without the use of a central manager [141]. In order to do so, P2P networks extend on the definition of overlay networks using a number of well-defined features. Such attributes, listed in Table 3.2 are elements that are compatible with SO design [142].

This chapter tackles this critical part of manufacturing infrastructure by defining a basic set of services that would allow applications and devices to join a survivable network of distributed nodes and participate in traffic relaying. Such a set consists of three services: networking, discovery and management. The first of these establishes reliable communication mechanisms for the transfer of messages between manufacturing infrastructure nodes. The second service is required to allow for the instantaneous detection of service advertisements throughout the enterprise. Finally, the management service is the orchestrator of all executed services on a node. Structuring the communication system in this way separates communication from any specific application or technology, such as OPC UA, while allowing them to participate and benefit from the system. Overall, this implementation is expected to improve the survivability of manufacturing systems in the enterprise.

This chapter is structured as follows. First, a discussion on the various possible overlay networking solutions is presented to justify the selection of P2P networks as

Layer	Properties
Application-level	Applications Tools Services
Services-specific	Meta-data Services management Services scheduling Services messaging
Features management	Security management Resource management Reliability and fault resiliency
Overlay nodes management	Routing and location lookup Resources discovery
Network communications	Network

Table 3.2: The typical architecture of P2P networks [143].

cooperative systems as the most appropriate technology for the communication system. Using the principles of the selected subdomain, a cooperative systems P2P protocol is designed, implemented, and evaluated experimentally. Next, the service-based architecture representing the networking, discovery, and management functions are described. This design is also implemented and evaluated experimentally. Finally, the chapter concludes with a discussion on the benefits and limitations of the system and gives some recommendations for future work.

3.1 P2P Networking

Generally speaking, the definition of a P2P network is a network in which a number of devices are connected together and share resources using direct exchange in a manner that is resilient against failures and transient population sizes, all without the use of a central manager. P2P networks are constructed upon existing physical infrastructure and, consequently, are normally considered to be a subclass of overlay networks. Currently, the largest use of P2P networks is file-sharing applications, however, they may be used to share content, storage or CPU cycles. The remainder of this section will delve into the difference between these classification types and their methods of operation [144, 141, 140].

3.1.1 Centralisation, Structure, and Hierarchy

P2P networks may be distinguished from one another based on their structure, topology, and degree of centralisation. As such, there exist centralised and decentralised networks, hybridised or partially decentralised networks, structured and unstructured networks, Horizontal Hierarchical (HoHA), Vertical Hierarchical (VeHA), hybrid hierarchical networks, and P2P networks as systems.

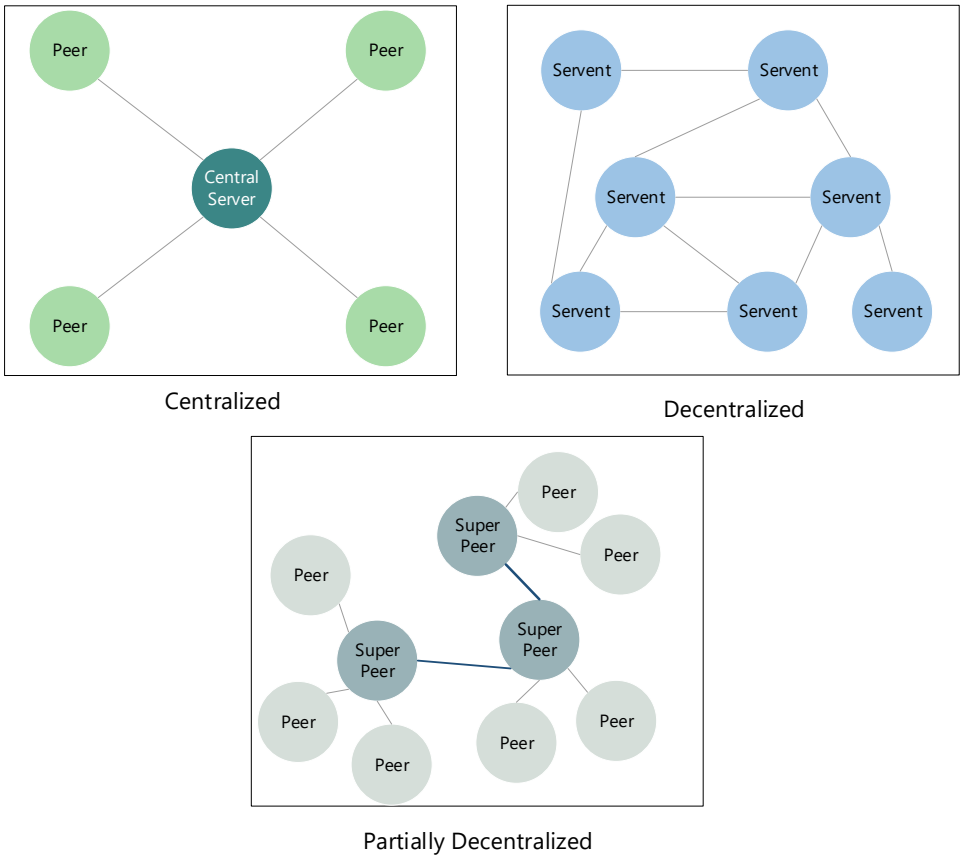

Figure 3.1: Representative examples of centralised (top left), decentralised (top right), and hybrid partially decentralised (bottom) networks.

Centralised networks, shown in Fig. 3.1, normally require central servers to manage metadata, exchange data, route search requests, and coordinate peer efforts. However, the central server does not share resources itself. The use of a central server introduces a SPoF and limits the size and reliability of the network [144].

Decentralised networks, also shown in Fig. 3.1, eliminate the central server and have peers handle all requests instead. If the architecture is purely decentralised, then each peer can behave as both a server and client [144]. These peers are termed servents as they are capable of occupying both SERVer and cliENT roles [145]. The Gnutella v0.4 protocol is a purely decentralised protocol composed of randomly connected servents that match and respond to queries (server role), generate queries, and consume subsequent query responses (client role) [146].

In a partially decentralised deployment, peers with higher abilities may be promoted to a more important role in the network. This role is referred to as 'super peer', as shown in Fig. 3.1. For example, Gnutella v0.6 networks operate by ensuring that 'leaf peers',

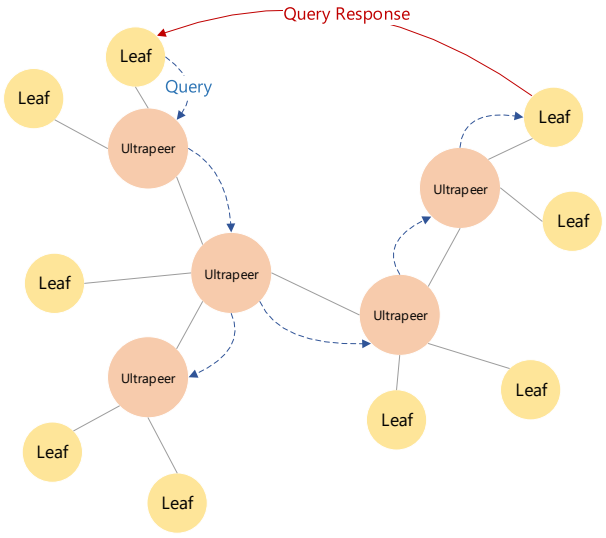

Figure 3.2: An unstructured Gnutella v0.6 network and searching algorithm.

or normal peers, are connected to a minimum of one 'ultrapeer', or super peer. The ultrapeers connect to each other forming a two tier network. Each ultrapeer manages a hash table of its leaf peers' resources and necessitates that all of its leaf peers' queries go through it. At the network level, this implies that only ultrapeers can forward messages. These attributes justify the description of ultrapeers as local servers, and, consequently, the network as a hybrid or partially decentralised network [144, 146].

Unstructured networks dissociate the location of the resource from the topology of the network. To locate a resource, queries are normally flooded, or forwarded using depth-first or breadth-first algorithms, until the resource is located or the queries expire. A representative example is the Gnutella v0.6 protocol, where its ultrapeers utilise a flooding mechanism, shown in Fig. 3.2, to forward queries amongst themselves in preference to a DHT based method. This factor alone causes the Gnutella v0.6 protocol to be defined as an unstructured network [146].

In contrast to unstructured protocols, structured ones map the available resources to their locations using, for example, resource identification codes and node addresses. Structured networks also normally use DHTs for routing. An example of a structured protocol is Kademlia which is also one of the most popular DHTs. BitTorrent, for example, currently uses a flavour of Kademlia for its network. Kademlia works by randomly assigning a Node ID from a 160-bit identifier space to each member of the network. The distance between two identifiers in the key space is determined by performing an exclusive or (XOR) on the two identifiers. Key-value pairs are then stored on the peers closest to an identifier. A binary tree with k-buckets as leaves covering the entire key space is then used as a routing table for lookups. Intricacies aside, Kademlia is capable

VeHA	HoHA
Network traffic load distribution using network heterogeneity	Homogeneous environment allows for per node load balancing
Versatile adaptability to various network environments; capable of withstanding the presence of firewalls and NAT	Aware of topological physical proximity
Intra-layer administrative autonomy	Full administrative control and autonomy
Decreased search latency	Efficient replication and caching

Table 3.3: A comparison of the advantages of VeHA and HoHA [140].

of resolving lookups in O(log(N)) hops, where N is the number of nodes in the network [144, 147, 148].

In Kademlia, a resource publisher would generate a hash for the resource to be shared. This hash then acts as the key in the key-value pair. The publisher's identifier is used as the value. The key-value pairs are kept on the peers with identifiers closest to the generated key, respectively. To retrieve a resource, a search is performed for the nodes with IDs closest to the resource hash key. From these nodes the list of sources may then be retrieved. Thus, DHTs may be created that are based on proximity and are resilient to failures and node churn. It is, however, important to note that these DHTs are vulnerable to Sybil attacks as there are no measures in Kademlia that prevent a node from generating several false identities [147, 148, 149].

Hierarchical networks, on the other hand, divide their members into layers based on their functions and capabilities. Hierarchical networks can be composed of structured, unstructured or both types of networks.

An example of an unstructured hierarchical network is the previously mentioned Gnutella v0.6. This is because the unstructured Gnutella v0.6 network has a partially decentralised topology with ultrapeers and leaf-peers working in two separate layers. This separation makes it an unstructured hierarchical network [146].

Structured hierarchical networks, on the other hand, can be seen as the way in which researchers attempted to combine the benefits of both structured and unstructured networks into single protocol definitions [140]. These networks can exist as vertical or horizontal hierarchical networks; abbreviated as VeHA and HoHA, respectively. VeHA networks are defined as ones where each layer of the hierarchy has its own DHT. HoHA networks have a single DHT for the entire network [150]. Consequently, VeHA networks require the use of gateway nodes to connect the separate layers while HoHA networks do not. A comparison of the advantages of VeHA and HoHA is shown in Table 3.3.

Finally, a hybrid hierarchical network uses a structured topology for one layer and an unstructured topology for another of its layers [140].

The field of P2P networks as systems is concerned with the bridging or merging together of networks to allow for expanded systems, inter-system content sharing, and inter-system traffic engineering. The first of these aims to group together nodes that share similar goals into a single system to eliminate the factor of competition between

them. Inter-system content sharing, as the name implies, has systems cooperating with each other to share resources. Finally, inter-system traffic engineering focuses on using P2P systems to optimise network routing performance [151].

Discussion

When comparing hybrid systems to purely decentralised networks, the former is found to reduce discovery time and inter-nodal traffic [145]. Hybrid systems also eliminate the SPoF drawback associated with centralised networks. Yet, they do so at the cost of slower discovery times [145]. In order to overcome this latency, indexing may also be distributed across the network as is done in structured systems.

Using a vertical hierarchical protocol, however, affords a system the advantages of partially decentralised networks. For example, VeHA allows for the integration of heterogeneous nodes with differing capabilities, and is capable of respecting these differences. Furthermore, VeHA protocols are able to integrate with the networking infrastructure found in manufacturing enterprises and adapt to routing controls, such as firewalls and NAT.

Likewise, the concept of P2P networks as systems affords participating infrastructure the flexibility to deploy separate overlays of similar or different protocols that may then be dynamically bridged, merged, or kept as separate to facilitate or restrict communication between them as necessary.

Thus, both VeHA and P2P networks as systems are viable options for the design of a resilient communication protocol for survivable manufacturing infrastructure. The next sections will detail each based on recent literature to determine which is more suitable. First, to establish sufficient background on the overall domain, traditional networks are discussed.

3.1.2 Traditional P2P Networks

The next sections will review VeHA P2P protocols and P2P networks as systems. Since these build upon traditional P2P protocols such as Chord, Tapestry, and Kademlia, this section first gives a description of the most popular traditional P2P routing protocols as a precursor to the coming discussions. This review of accepted protocols also serves the development of the transport protocol in the second half of the chapter by, for example, providing viable mechanisms for routing requests, joining networks, and network discovery.

This section examines protocols for both structured and unstructured networks. Thus the Chord, Tapestry, Content Addressable Network (CAN), Pastry, Kademlia, and Viceroy structured networks are described. The Freenet, Gnutella, and BitTorrent unstructured networks are also discussed. As Kademlia and Gnutella have already been covered in Subsection 3.1.1, they will not be presently discussed again.

A comparison of the structured P2P protocols covered in the next section is presented in Table 3.4. The comparison is made based on parameters partially sourced from [143] and defined in Table 3.5.

	Chord	Tapestry	Pastry	CAN[1]	Kademlia	Viceroy	P-Grid
Centralisation	Decentralised	Decentralised	Decentralised	Decentralised	Decentralised	Decentralised	Decentralised
Architecture	1-dimensional, unidirectional circle of 2^m points known as the Chord ring.	Plaxton Mesh	Plaxton Mesh	d-dimensional coordinate space	160-bit keyspace treated as a binary tree.	Approximate butterfly network and connected ring.	Binary trie
Bootstrap Requirements	Knowledge on any existing node. Stabilisation protocol.	Node insertion protocol.	Knowledge on a nearby existing node based on the proximity metric. State initialisation protocol.	DNS system. Bootstrapping domain name. One or more bootstrap nodes.	Knowledge on any existing node.	Presumably, knowledge on any existing node capable of propagating a successor lookup.	Bootstrap node.
Node Exit Procedure	Each node keeps a list of successors r. If a successor fails, the next one in the list is used.	Exit is announced. Replacement nodes are identified and informed.	No protocol for graceful exit.	Explicit handover of node's zone to a neighbour.	No protocol for graceful exit.	Remove all outbound connections. Inform all connected servers of intention. Transfer resources to successors	No protocol for graceful exit[2].
Lookup Protocol	Key value pairs. Key is mapped to a node on the Chord ring. Successors for forward routing. Finger tables are used to scale lookups.	Key-identifier match suffix routing	Key-identifier match prefix routing	Key value pairs. Key is mapped to a point in the coordinate space. Greedy forwarding to the neighbour with coordinates closest to the destination coordinates.	Key value pairs assigned to nodes using XOR between identifiers. Lookups follow a recursive algorithm of parallel and asynchronous queries starting from a number of closest nodes.	Key mapping identical to cord. Multi-phase routing traversing the butterfly, first up, then down and sideways, and finally along the ring.	Key value pairs mapped to keyspace partition. Key-identifier match prefix routing.
Routing Performance	$O(logN)$	$O(logN)$	$O(logN)$	$O(d.N^{1/d})$	$O(logN)$	$O(logN)$	$O(logN)$

Security Features		None	Architecture supports secure channels, node authentication and message authentication codes.	API supports node authentication	None	Resistant to a DoS attack using bulk node-joins.	None	Node identity verification using public keys. Trust and reputation management.
Resilience	Node Joins	Large number of joins may cause lookup delays.	Four component node insertion process for routing table consistency and object availability.	Optimistic approach to Node arrival affects a small number of nodes.	Joins are localised and only affect $O(d)$ existing nodes.	A new node is inserted in a capped bucket only if the least-recently seen node is unresponsive.	[152] assumes multiple joins do not overlap.	As demonstrated in [153], network functions are unaffected by network mergers, which is an operation involving a high level of join operations.
	Node Failures	Increasing r increases system robustness.	Multiple object root peers and backup neighbours.	Lazy (opportunistic) repair of routing table. Incorrect entries persist until they are discovered by being used.	Repair using expanded ring search and immediate takeover algorithms. Key-value pairs are lost until the state is refreshed by data holders.	Node state exhibits preference to oldest living nodes. Incorporates replication.	[152] assumes multiple leaves do not overlap and servers never fail.	Redundant nodes per keyspace partition.

Symbols: m is the length of hash identifiers. N is the number of nodes in the network. n is the size of the Tapestry namespace. Node IDs are in base b. d is the number of dimensions.
[1] The CAN base protocol is reviewed excluding the possible design improvements discussed in [154]
[2] There is no mention of a procedure for gracefully leaving a network in associated publications. Based on a review of the P-Grid code from http://www.p-grid.org/download/files.html, a leave() function is defined. Tracing the function to the CoUPolicy.java file in the pgrid.core.maintenance.identity package shows that the function is empty and does nothing.
For more performance metrics please review [155]

Table 3.4: A comparison of structured P2P protocols [143, 156, 157].

Parameter	Definition
Centralisation	Degree of centralisation in the network. The options are centralised, decentralised, and partially decentralised modes of operation
Architecture	Operating design of the resulting network
Bootstrap Requirements	The elements used in integrating newly joined nodes
Node Exit Procedure	The process executed on the network when a node leaves
Lookup Protocol	The process used for queries in the system
Routing Performance	Evaluation of query traversal procedure in the overlay
Security Features	The components integrated in the network to counter security threats
Resilience	The network's resilience based on the network's node exit procedure

Table 3.5: Definition of the parameters used for P2P protocol comparisons, some of which are sourced from [143].

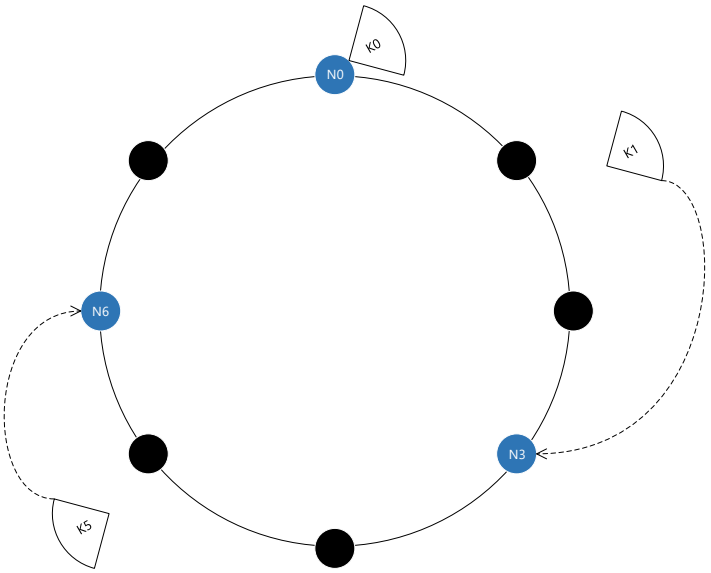

Figure 3.3: A chord ring with m=3, 3 nodes, and 3 keys. Keys K1 and K5 are located at nodes N3 and N6 as they are the successors to nodes N1 and N5, respectively.

Structured P2P Networks

The Chord P2P protocol is primarily built on consistent hashing. Consistent hashing is a method designed to minimise the disruption felt in a network as nodes enter or leave a network. It attempts to equally distribute the keys between the network members such

that each node maintains a routing table on O(logN) network peers. This allows for a graceful degradation in performance as the validity of information expires. In more detail, the consistent hash function is used to assign m-bit identifiers to peers and data keys. These identifiers are generated for peers and data keys by using the SHA-1 function to hash the peers' IP address and the data key itself, respectively. Therefore, 'm' must be large enough to prevent the hashing function from generating identical identifiers for different resources. Identifiers are then mapped on a circle of size 2^{m} known as the Chord ring. A key is assigned to the first node that equals or follows the key in the identifier space, as shown in Fig. 3.3. The next node on the identifier circle is then known as its successor and if the first node leaves the network or fails, then its keys are assigned to its successor. This method ensures stability in the network in the face of node failures. Similarly, lookups are also dependent on the use of successors. That is, since it is required that each node know its successor, lookups traverse along these successor pairs until the desired identifier is located. Once the required node is found, the response travels back by reversing the path used to find it. A benefit derived from the use of successors is that each node needs to know only a small portion of the routing table. Finger tables are also used by nodes to scale lookups. A finger table consists of m entries, where the k^{th} entry is the first successor node that is at least $2^{\mathrm{k-1}}$ successions away. Thus, a node forwards lookups to the highest predecessor key in its finger table. This allows for $O(logN)$ lookup times. However, this high dependence on successor lookups means that routing information must be consistently accurate. To do so, Chord requires the use of a stabilisation protocol. This protocol is run periodically on each node to update the successor points and the finger table [143, 156, 158].

In contrast to Chord, Tapestry is highly focused on the routing aspect of P2P networks. Tapestry is founded on the methods proposed by Plaxton *et al.* in [159]. In [159], the Plaxton model defines three possible roles for nodes: servers, routers, or clients. These nodes then generate a distributed data structure known as the 'Plaxton Mesh'. Names in the mesh are essentially bit-sequences, that are random, fixed length, and unique in nature. Each object is then rooted at the node with the name closest to the object's name. Consequently, each object has one root peer. In contrast, each object in Tapestry has multiple root peers for redundancy [143].

In terms of routing, Tapestry also bases its methods on those of Plaxton *et al.* in [159]. Consequently, routing is done by matching Node ID suffixes. This involves the use of the destination's ID to route messages across overlays through digit-by-digit matching. Consequently, the routing table employed by peers consists of multiple levels, with each level corresponding to a suffix that matches to a certain digit position in the ID space. Tapestry does, however, introduce a modification to the methods of [159] by making the routing table take distance into account, and by including pointers to what are known as neighbour nodes [143]. An example of Tapestry routing can be seen in Fig. 3.4.

Tapestry also includes dictates on the management and reachability of data objects. For semantic flexibility, Tapestry stores the locations of all data object replicas and allows for the selection of a specific one based on a variety of criteria. For resilience, as previously mentioned, each data object is assigned multiple roots. Root selection

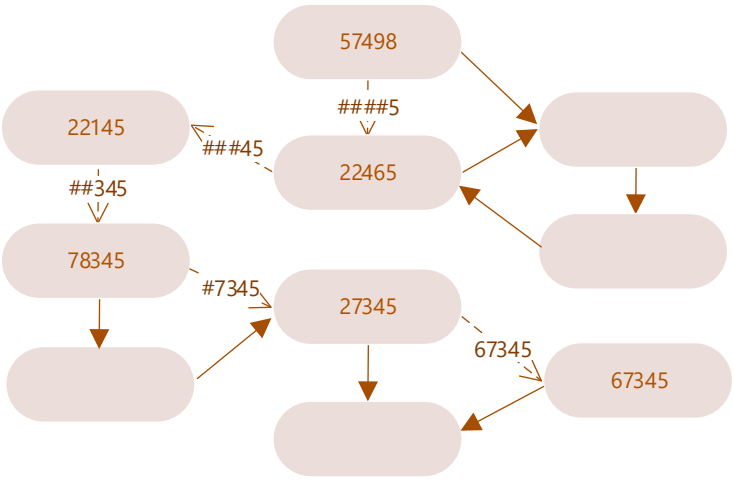

Figure 3.4: An example of Plaxton-based mesh routing in Tapestry.

therefore occurs through the use of surrogate routing, which is a form of routing that allows any identifier to be mapped to a peer in the network. Tapestry also enhances its fault tolerance features through the use of TCP timeouts and UDP heartbeat messages to detect faults. If a fault is detected, then each node contains entries for a primary neighbour and two backup neighbours in order to ensure the resume of normal operations [143].

The CAN network uses a Cartesian multi-dimensioned coordinate space on a multi-torus that acts like a hashtable. The entire coordinate space is partitioned such that every node has its own zone. Each node maintains a routing table that contains the IP addresses and virtual coordinate zones of its neighbours. The routing of messages towards a destination is then done using a greedy forwarding algorithm that selects the next hop as the neighbour that is closest to the destination coordinates. Key-value pairs are stored in the virtual coordinate space by mapping the key to a point in the space using a hash function. Consequently, by using the same function, any node may retrieve the value associated with a key [143, 154].

When a new node joins the CAN network it must perform three actions. First of all, the node resolves a CAN domain name to acquire the IP address of one or several bootstrapping nodes. Once contacted, the bootstrap node provides the new node with the IP addresses of several randomly selected peers that exist in the network. Using this information, the new node sends a 'join' request to a randomly selected point P. Once the node in zone P receives this request, the zone is split in two, as shown in Fig. 3.5. The newly formed zone with its associated key-value pairs and IP addresses of its neighbours are then assigned to the new node. The remainder of the neighbouring nodes then update their coordinates of the new zones and the associated IP addresses

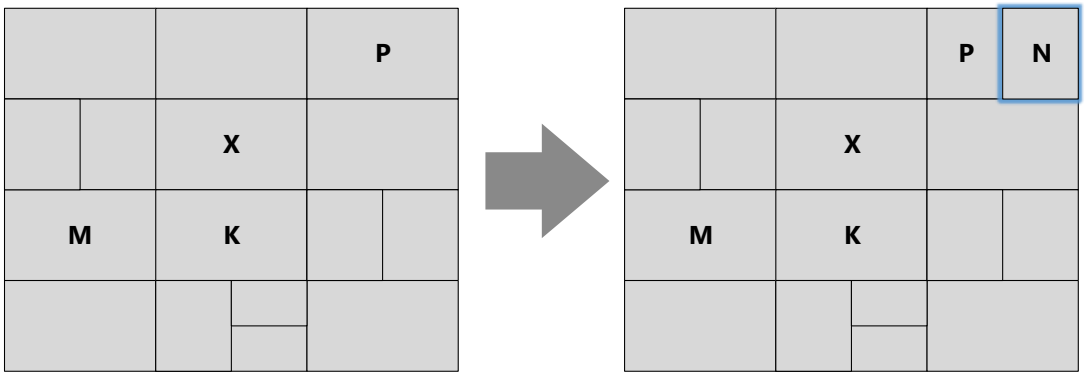

Figure 3.5: A 2D CAN network showing a split of zone P to accommodate a newly joined node N.

[143, 154].

When a peer fails, a takeover algorithm is used to assimilate the failed node's zone with the zone of a neighbour. The takeover node then updates its table to remove from it any peers that are no longer its neighbour. Soft-state updates are then transmitted throughout the system to ensure that all nodes are able to calculate if they now have a new neighbour, and to update their tables accordingly [143, 154].

Pastry, similar to Tapestry, uses prefix routing based on the methods of Plaxton *et al.* described in [159]. Each node is assigned a randomly selected 128 bit node identifier for the even distribution of nodes in a circular identifier space ranging from 0 to 2^{128} - 1. Pastry requires that each node maintains a routing table, a neighbourhood set, and leaf set, as shown in Fig. 3.6. The routing table consists of $\log_B(N)$ steps, where $B = 2^b$ and b "involves a trade-off between the size of the populated portion of the routing table (approximately $[\log_{2^b}(N)] \times (2^b - 1)$ entries) and the maximum number of hops required to route between any pair of nodes ($[\log_{2^b}(N)]$)" [160].

Each row of the routing table contains $B - 1$ entries. The node ID and key are considered sequences of digits in base 2^{b}. An entry in row n signifies that its node ID shares n digits with the current node. Each entry contains the IP address of the respective peer node. Pastry routes messages using the routing table by selecting the node with the node ID matching the prefix of a given key by an extra b bits than the current node, and that is chosen out of the available alternatives based on a proximity metric. If no such node exists, then a node matching the same number of digits as the current node, but is numerically closer to the key, is selected instead. Routing therefore usually takes less than $\log_B(N)$ steps [160].

The neighbourhood set for a peer, denoted by the symbol M, contains information on the $|M|$ peers closest to it in terms of proximity. As is the case for the routing table,

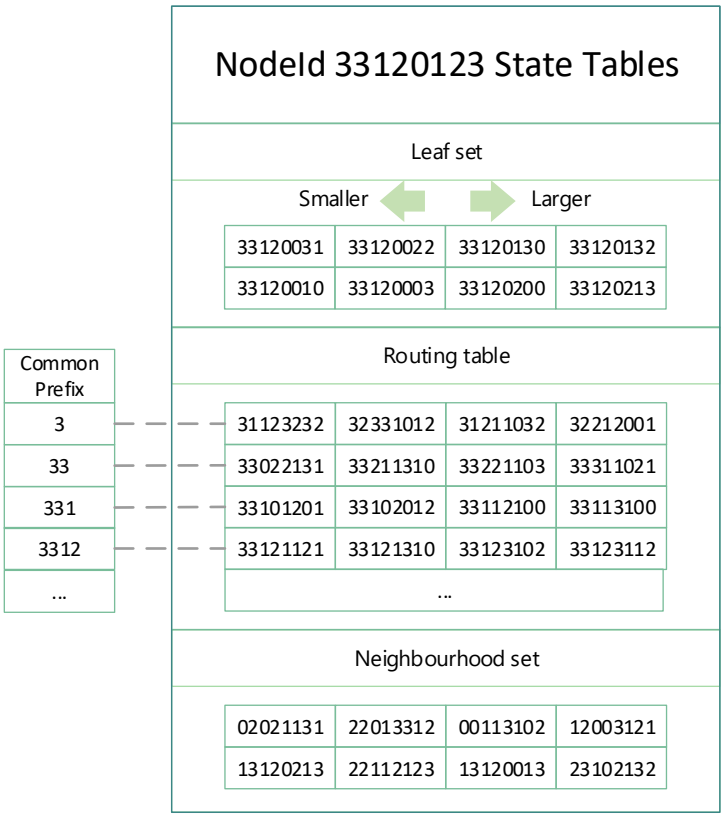

Figure 3.6: Pastry leaf set, routing table, and neighbourhood set. The host node has a node ID 33120123, $b = 2$, and $l = 8$. All IDs are in $base = 2^{\mathrm{b}} = 4$. Each row n in the routing table shares n digits with the host node.

entries consist of node IDs and their corresponding IP addresses. This set is not typically used for routing, but is meant to be used as a source of locality information [160].

Finally, the leaf set of a peer, L, is composed of the nodes with $|L/2|$ numerically larger and $|L/2|$ numerically smaller node IDs as compared to the peer's own node ID. The leaf set is used for message routing before the routing table. This means that if the key is within the leaf set's range, then the message is routed to the node with the node ID closest to it, which may be the current node. Otherwise, the routing table is used in the same manner as described earlier. Both $|L|$ and $|M|$ are typically set to 2^b or 2×2^b [160].

For a node to join a Pastry network it must initialise its tables and then announce its presence to the network. To initialise its state tables, the node must have the address of a contact peer to which it sends a 'join' message with a key equal to its node ID. The join request is forwarded to the node numerically closest to key. All the nodes that

encounter the join request forward their state tables to the new node. The new node may request more data from other nodes and announce its presence to nodes that need to be informed of its presence. The contact peer's neighbourhood set is used to initialise the neighbourhood set of the new node. Likewise, the last hop in the path of the forwarded "join" request forwards its leaf set to initialise the new node's leaf set as it is considered to be the numerically closest peer. Lastly, the new node communicates its state tables to its new neighbours for them to update their own tables based on this new information [160].

When a node fails in a Pastry network, and no longer responds, the leaf set, neighbourhood set, and routing table may need to be repaired. In the case of leaf set repair, the failed node's neighbour contacts the peer that has the largest index on the side of the failed node. The neighbour requests its leaf set and selects from it an appropriate node that is not already in its set. The selected node is contacted to ensure that it is operational before it is included. Guarantees for leaf set repair do not hold in the unlikely case where $|L|/2$ nodes with adjacent node IDs fail at the same time [160].

To repair the routing table of a node after a peer's failure, the node contacts a peer X from the same row as the failed node in the routing table. If the failed peer occupied entry R_l^d, where l is the row and d is the column, then the node requests this specific entry from node X. If none of the entries in row l are alive, then the node requests entries from the peers of row $l+1$ until an appropriate replacement is found [160].

Finally, to repair the neighbourhood set, the affected node requests that the remaining members of its set respond with copies of their own sets. The proximity of peers in the new sets are checked and appropriate peer(s) is(are) selected and inserted in the node's neighbourhood set [160].

The Viceroy protocol, like Chord, uses consistent hashing for data distribution. A DHT is used to manage this distribution, and to allow peers to contact servers and locate resources by name. Architecturally, Viceroy is composed of an approximate butterfly network and a ring of connected predecessors and successors. This means that each server has five links: three outgoing for long-range destinations, termed up, left and right links, and two ring links, one to its successor and another to its predecessor. [161, 152].

The procedure used by Viceroy to ensure the balanced dispersion of nodes and data across a network is used and executed by newly joining nodes. A new node performs an estimation on the total number of active nodes in the network, n, using local data once it joins the network. With this information in hand, an architectural level is selected uniformly from the set of $[1...log(n)]$. The node also selects a 128 bit identifier uniformly from the range $[0, 1)$. If a clash in IDs between nodes is detected later on then the clash is resolved by adding multiple precision bits to the ID. Resources that lie within the range [*predecessor-id...node-id*], where the node-id is the newly joining node's identifier, are requested by the new node from its successor. Once the handover is complete, the successor purges those values from its stores. The opposite of this algorithm occurs when a node fails or leaves the network [161].

Viceroy is compatible with five different lookup algorithms. These are the Paper,

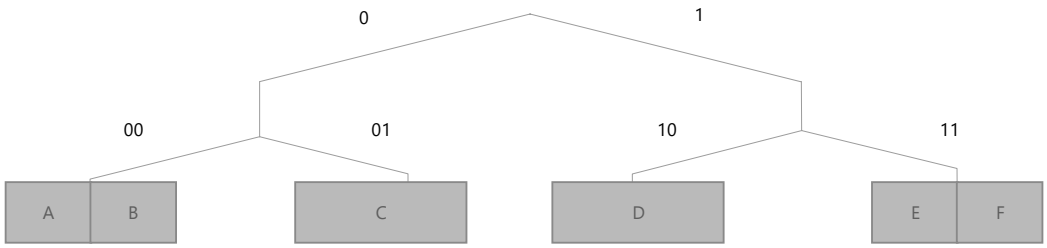

Figure 3.7: An example P-Grid.

PapersPlus, TraverseTreePlus, TraverseTreeAbsolute, and FindFast algorithms. The first of these algorithms, Paper, forms the basis for PapersPlus, TraverseTreePlus, and TraverseTreeAbsolute so that they are, in effect, extensions to the Paper protocol [161].

The Paper algorithm consists of three different phases: the 'Proceed to Root', 'Traverse Tree', and 'Traverse Ring' phases. In the first phase, the lookup is routed to the root using up links. The second, the 'Traverse Tree' phase, has the query travelling the tree using either left or right links until the query overshoots its destination or the required son is determined as non-existent. Finally, the third phase has the query traverse the ring using successor and predecessor links until the node, and therefore the resource, is found [161].

The PapersPlus algorithm is an extension to the Paper algorithm. It adds an additional check to the 'Traverse Tree' phase such that it determines which of the two child nodes (left or right) is responsible for the value being looked up. Once this is determined, the 'Traverse Ring' phase is initiated using the selected child [161].

The TraverseTreePlus algorithm introduces a modification to the 'Traverse Tree' phase. Here, if a child is determined to exist, then the query is either routed to the child or the 'Traverse Ring' phase is initiated [161].

The TraverseTreeAbsolute protocol also modifies the 'Traverse Tree' phase. It requires that the child closest to the target be selected before the 'Traverse Ring' phase is initiated, regardless of whether it is the left, right, or current node [161].

The final algorithm, FindFast, uses greedy searches to locate the target node and resource. In searching for targets, the protocol uses absolute distances to select the closest node. This means that the FindFast algorithm is completely structure-agnostic and does not use the Viceroy's architecture in any way [161].

Lastly, the P-Grid protocol, like Pastry and Tapestry, uses prefix-based routing. A distributed binary search tree is constructed with one or more peers responsible for a branch of the tree. For example, Fig. 3.7 shows that node C is responsible for the path 01, thus it stores all data with keys having a prefix of 01. For fault tolerance, several peers can be responsible for the same path, as is the case for nodes A and B in Fig. 3.7. Unlike in Pastry/Tapestry, node identifiers and routing paths are independent of

each other. This allows paths to change as required to allow for the equal distribution of storage load in the network. These changes are done dynamically as part of the maintenance protocol. It's worth noting that sampling-based algorithms are used for the balanced replication of resources [162].

For query routing, P-grid necessitates that each peer at a level store a reference to another peer at the same level but at the other side of the tree. Node A in Fig. 3.7, for example, would store a reference for path 01 pointing to peer C, and path 1 pointing to peer D. The search cost remains logarithmic with dynamic paths because keys are resolved in blocks and not in a bit-wise manner. Data updates, on the other hand, are announced in a network using a rumour spreading algorithm based on a push/pull gossiping scheme [162].

A public key distribution mechanism similar to PGP is used with a "quorum-based query scheme" for the verification of node identities [162]. The public keys are stored in-network for verification guarantees similar to the web-of-trust. The modelling and analysis of peer interactions is also used to determine and assign trust ratings to peers through a voting scheme to allow for the identification of unacceptable or malicious behaviour [162, 163].

Unstructured P2P Networks

The Freenet network is a distributed file storage and retrieval system focused on anonymity and availability. For a node to join the network, the address of an existing node should be known before-hand. Each node maintains a routing table with its data keys and the addresses of network peers, as well as a locally set value for disk space assigned to host shared files. To ensure privacy for its members, nodes only know their immediate upstream and downstream neighbours. Consequently, similar to IP, queries are routed via a chain of proxies with each node deciding on the most appropriate immediate next hop. Likewise, a Hops-to-Live (HTL) counter is used and decremented with each subsequent hop mirroring the Time-to-Live (TTL) measure of IP. Request identifiers are also used to prevent routing loops, again, similar to IP [164, 143].

To retrieve a file, a request is sent with the HTL and the binary key file. If a node receives the request and has the file, the file is returned and signals that it is the data source. Otherwise the request is forwarded to the node in its routing table with a key closest to the requested key. If that node is down, then the node with next closest key is selected. This process repeats as many times as required. Requests that make their way back to a previously visited node are dropped to avoid routing loops. A successful request results in the file being cached by the forwarding nodes on the return path, as shown in Fig. 3.8. Requests with the same key can then be resolved by the local cache. Requests with similar keys are forwarded using the same path. These two features should improve the effective routing and success of requests. If a successful response is received from a node not already in the routing table, it is added thereby supporting network discovery. To maintain the local cache within storage limits, the least used files are evicted when a new file is received that would exceed the cache size. Additionally, nodes on the return

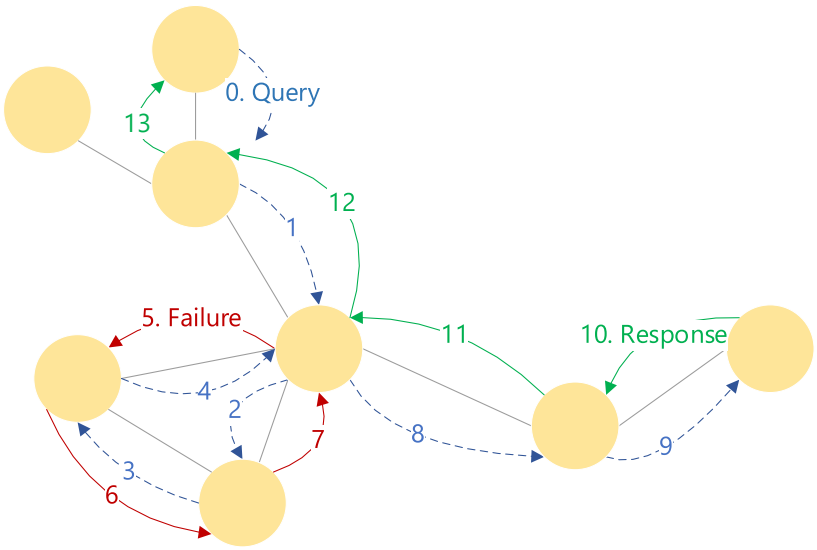

Figure 3.8: A request lookup in the Freenet network. A failure is returned in step 5 to break a routing loop causing the request to be forwarded down an alternate path in step 8.

path may arbitrarily declare themselves as the data source to maintain the privacy of the original source [164].

Data keys in the Freenet network are normally generated using a 160-bit hash function. The methods used to generate the keys may differ. The Freenet protocol may produce binary key files using one of three methods: the Keyword-Signed Key (KSK), Signed-Subspace Key (SSK), and the Content-Hash Key (CHK) methods [164].

The KSK approach is the simplest one of the three used to generate a file key. KSK applies the hash function to a descriptive text string used to identify the file. This generates a public-private key pair. The public key is hashed to create the data file key and the private key is used to sign the file. The file is also encrypted using the text string. Thus, to share a file, the descriptive string should be published. It is important to note that the dependence of KSK on a descriptive text string leaves it open to dictionary attacks and 'key-squatting'. The former has a person hashing a list of possible descriptive strings to illicitly gain access to the resources. Key-squatting, on the other hand, is when a user publishes a file using the same descriptive string as an existing resource. These vulnerabilities are addressed by SSK [164].

SSK's enhanced security measures are based on the use of personal namespaces. SSK requires that each user generate a random public/private key to identify the namespace. To generate file keys, the public key and the descriptive strings are hashed independently, XOR'd together, and hashed once again. As is the case with KSK, the file is signed using the private key and encrypted using the descriptive string. To share a key, however, SSK

requires that both the public key and the descriptive string be published [164].

The CHK algorithm is focused on updating and splitting. A CHK of a file is generated by hashing its contents. Files are encrypted using random keys. To share a file, the CHK and encryption key need to be published. To allow file updating, the CHK method may be used in conjunction with SSK. To explain, a node adds a file to the network using CHK, but publishes the CHK using SSK. This allows file updates as the node may replace the file and the CHK and add a new indirect file to the SSK pointing to the new file. Any node with the old version receiving an insert of the new one will note that the latter is both valid and more recent and will therefore use it to replace the older file. In the case of file splitting, a CHK is divided into smaller ones and an indirect file is generated and used to point to the different divisions [164].

BitTorrent is a centralised P2P protocol that is also the most commonly used protocol. The protocol relies on metainfo .torrent files. Torrent files contain information on the shared resource and are hosted on web servers. Each .torrent file also contains a URL pointing towards its tracker. The tracker is a server that manages the list of nodes that form the active swarm of peers sharing the resource in question, as shown in Fig. 3.9. The tracker is normally also aware of the download progress of each member of the swarm. Any peer that has a full copy of the content is referred to as a seed [165, 166].

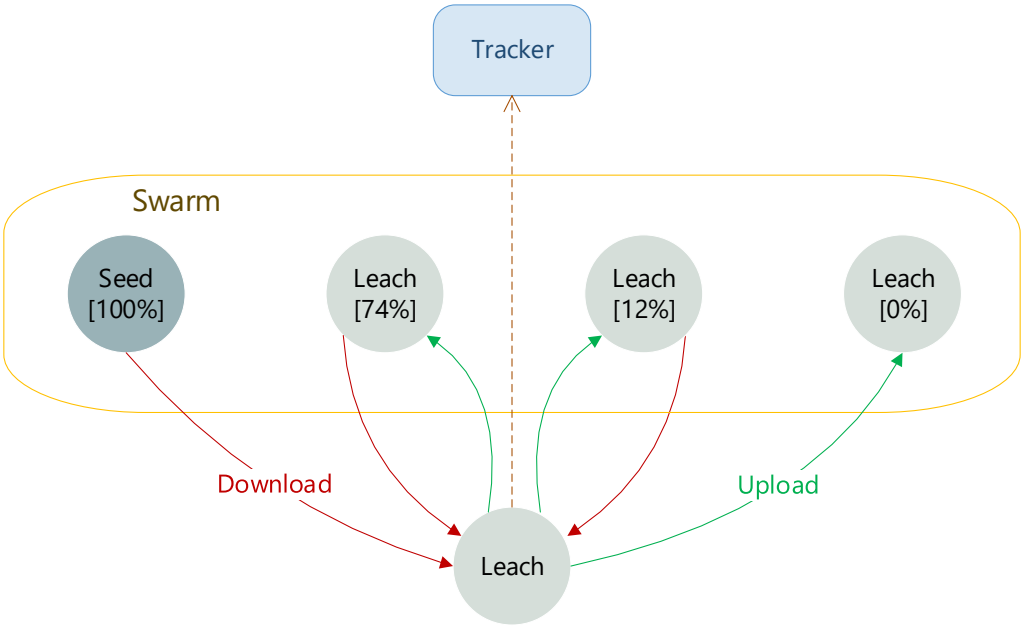

Figure 3.9: An example BitTorrent system. A new leach contacts the tracker to receive a list of peers from the tracked swarm. The leach connects to the swarm and downloads/uploads pieces of the shared file(s).

The BitTorrent protocol defines two piece selection strategies and four methods of peer selection. The former is composed of the random first, rarest-first, strict priority, and endgame mode approaches while the latter comprises of tit-for-tat, optimistic unchoking, upload only, and anti-snubbing.

Piece selection depends on the splitting of files into smaller pieces. The pieces are shared between nodes and reassembled when a complete set is acquired to recreate the original file. In the rarest-first algorithm, a node at the start of a download reviews the distribution of pieces throughout the network and selects the rarest piece to download first. This method is targeted at information survivability by trying to keep a full copy of the file in the network in case the last seed leaves before any other node completes its download. This approach also improves throughput as it diversifies the selection of pieces distributed across nodes. In contrast to rarest-first, random first, as the name implies, selects a random piece to download. This allows a node to quickly acquire its first piece(s) and contribute back as an uploader as soon as possible. A random first node usually switches to rarest-first after acquiring a small number of complete pieces. The strict policy algorithm works by having all the blocks of a piece prioritised if a single block from that piece is downloaded. Finally, end game mode is when a peer receives requests for all pieces or has outstanding requests for all pieces. At this point, the node requests the blocks it has not yet received from all nodes having them. Any block that is then received causes the node to cancel all its requests for the same block [167, 168].

As for peer selection, these are based on the concepts of choking and unchoking. A peer is said to choke another if it refuses to upload to it. The lifting of this ban is known as unchoking. The tit-for-tat mechanism requires that a peer unchokes up to a maximum of 4 peers at any given time. This is usually the 4 nodes with the highest upload rates to the node. Optimistic unchoking periodically selects a random node to unchoke regardless of its upload rate. This is done to allow for the discovery of new, possibly better, peers. Upload only mode is set by nodes that have completed their download. Like tit-for-tat, upload only mode selects peers with the highest upload rates to allow for the quick replication of the resource being shared. Finally, anti-snubbing is when a node, being choked by all of its peers, stops uploading to them [166, 168, 169].

3.1.3 VeHA Networks

This section will give an overview of VeHA networks by reviewing a number of solutions from literature [170, 171, 172, 173, 174].

In [170], a two layer network named brocade is proposed. Brocade is motivated by the desire to reduce latencies in routing using knowledge on the capabilities and locality of peers in different administrative network domains. This information is used to designate nodes in existing infrastructure with high bandwidth, processing power, and accessibility features as "supernodes". Each collection of peers in a domain or subdomain is grouped as a "cover set" and assigned to the most local supernode, as shown in Fig. 3.10. Supernodes then become the dominant entry points for the different network domains and subdomains. Thus, brocade is based on the assumption that knowing the

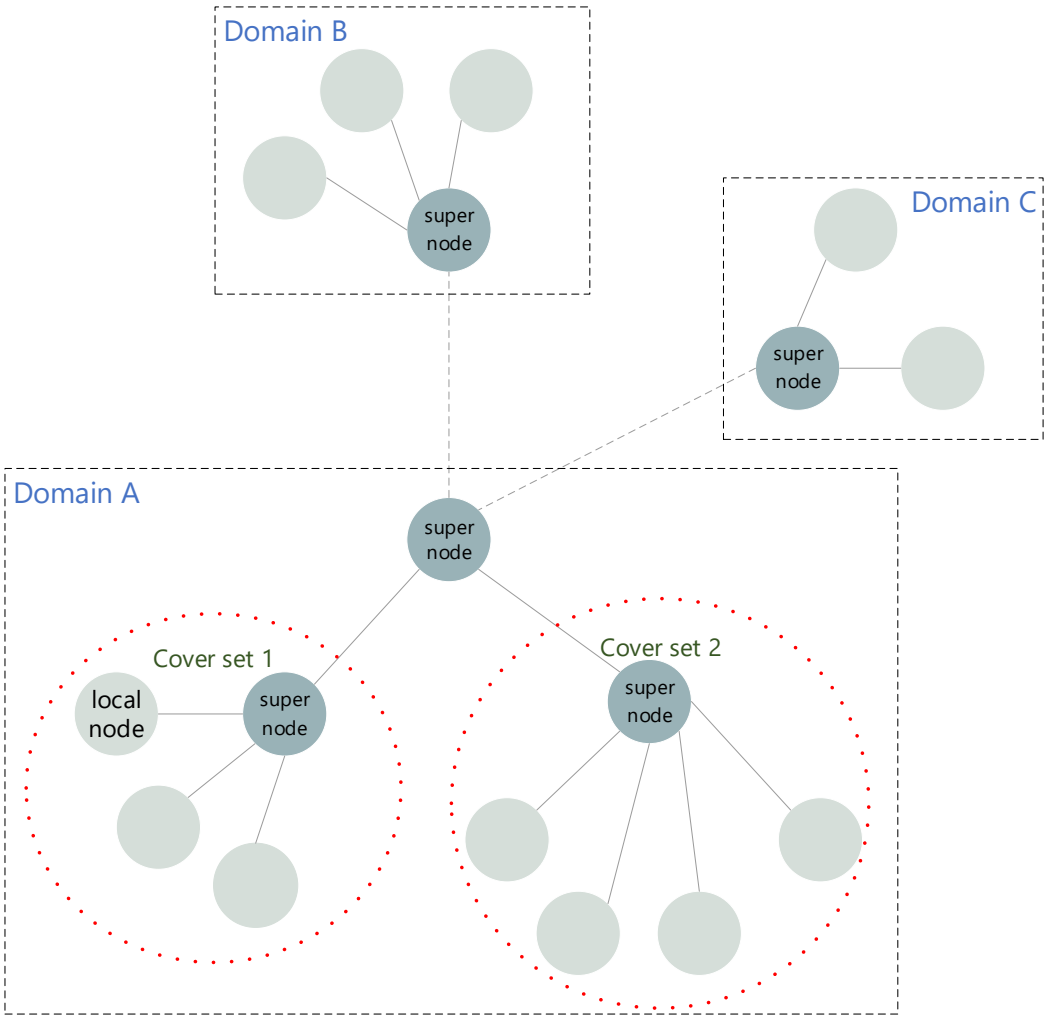

Figure 3.10: An example of a brocade network hierarchy.

network domain of the destination node and using the supernode overlay to route directly to that domain would result in a drop in hop count, bandwidth use, and network latency.

To demonstrate key routing mechanisms, a brocade is developed in [170] based on a Tapestry network. The concepts addressed are the differentiation of inter-domain from locally routed messages, the discovery of local supernodes in subdomains, and the discovery of destination supernodes in message routing.

Message filtering is resolved by having each supernode keep a list of the Tapestry nodes in its cover set. The number of entries in each list is assumed to range in the order of 10^4. Each supernode uses its list to determine if the destination of a message is inside

or outside the cover set. This allows supernodes to infer if brocade routing is applicable to the transiting message.

As for the discovery of local supernodes, naive, IP-snooping, and directed approaches are proposed. In the naive approach, an inter-domain message is routed through the brocade overlay if it happens on a supernode as part of its Tapestry-outlined path. Thus, there is no guarantee that an inter-domain message will take advantage of the brocade overlay.

IP-snooping, on the other hand, assumes that supernodes are more likely to be the nodes at the edges of network domains. Therefore, it infers that messages destined for external nodes will probably transit through them. IP-snooping therefore requires that supernodes inspect IP packets and parse the message headers to determine if they are Tapestry messages with an external destination ID. If both conditions are met, then brocade routing may ensue. This method is considered to be difficult and costly to implement.

Directed discovery uses DNS resolution or an expanding ring search to determine the local supernode. Each node also keeps a "proximity cache" on the peers it previously communicated with. Nodes in the proximity cache are assumed to be within the same cover set. Messages destined to these nodes are routed directly. Otherwise, the message is sent to the supernode for routing. State is maintained through the use of periodic beacons. Aside from the cost of state maintenance, this method would also require fault tolerance at the supernode level. It is however stated as the best option of the three [170].

Finally, the determination of the destination supernode is achieved by created a second Tapestry overlay for the supernodes. Thus, Tapestry routing using the node ID suffix is used to route the outgoing message to the destination supernode, which would then route the message to its final destination within its cover set.

Brocade is evaluated in [170] through simulations and is shown to achieve its outlined goals of reduced bandwidth consumption and improved routing performance.

In [171], a general framework is presented for hierarchical DHTs based on the hierarchical nature of the Internet. Initially, the protocol requires that nodes be organised into disjointed groups. For a member of a group to communicate a lookup message to a destination, the message is first routed to the destination group and then to the destination using inter-cluster and then intra-cluster routing, respectively. Like in [170], super-peers are selected from the pool of available peers to form a secondary overlay. The selection primarily occurs based on the reliability and connectivity characteristics of the peers. Criteria such as bandwidth and processing power may also be taken into account. The secondary overlay is responsible for inter-cluster routing. Consequently, every cluster must have at least one super-peer. It is important to note that the proposed hierarchy is not limited to two levels, but may be expanded to accommodate as many layers as required. In doing so, a lookup message would have to first be routed to the super-peer of the highest layer, which would then hand it off to the super-peer of the level below it and so on until the message is routed within the destination's group and to its destination. The proposed framework is evaluated using Chord as the top-level

DHT for a two-layer network. The results observed showed a substantial decrease in the number of hop counts traversed by lookup messages. Based on the description of the general framework, [171] can be considered as having a high degree of similarity with [170].

In [172], a framework termed HIERAS is proposed for the division of clusters into connected sub-clusters, termed P2P rings, to improve the routing performance of DHT-based networks. The protocol requires that all peers be part of one all-encompassing DHT. Within the network, nodes are split to form sub-clusters based on topological proximity, as shown in Fig. 3.11. If required, the nodes of a sub-cluster may be split again into micro-clusters as many times as required. Each peer must therefore be part of a cluster at each layer, and, as such, all peers are equal. There are no super-peers or gateways in this framework. When a routing task is to be performed, it is first tackled at the lowest layer, within the smallest sub-cluster of which the node is a member. Thus, tasks are resolved locally when possible, and routes are composed of low latency hops.

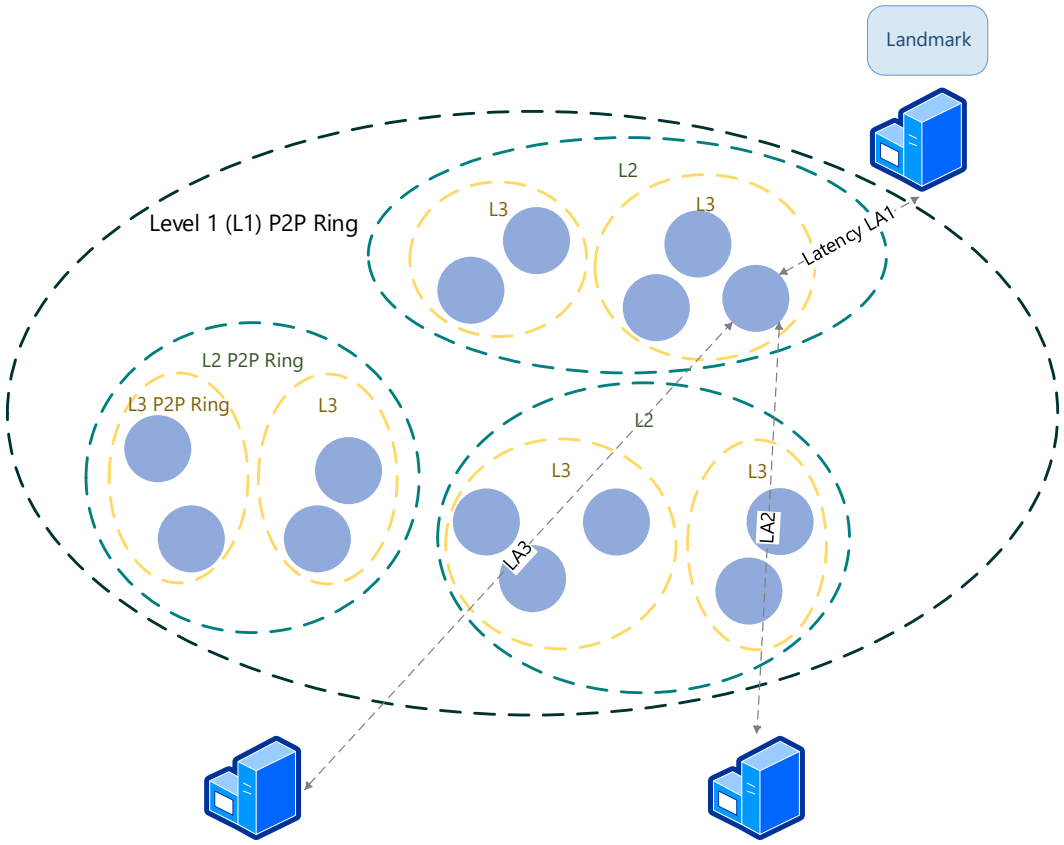

Figure 3.11: An example of a three level HIERAS network hierarchy.

This results in a shorter overall latency in routing and also alleviates the strain of routing from the upper layers.

The P2P rings in [172] are formed using a distributed binning mechanism described in [175]. Several endpoints are selected from the Internet to serve as landmark nodes. These are well known machines that are spread out across the Internet. The nodes in a HIERAS network divide themselves into disjoint bins based on their proximity using the network link latency between themselves and the landmark nodes as a reference, as shown in Fig. 3.11. Thus, aside from the routing tables of the underlying algorithm, nodes must also maintain a landmark table and ring tables. A ring table is labelled by ordering the measured latencies to landmark nodes. It is kept on the node with the node ID closest to the ring table's label. The ring table is also replicated on several other nodes for fault tolerance. Typically, it contains the smallest two and largest two node IDs of the nodes participating in the P2P ring. The ring table host maintains the table by periodically checking the status of its constituents.

An increase in the hierarchical depth in HIERAS improves the routing performance. This is at the cost of higher overhead as more state information and ring maintenance operations will be required. The evaluation of the system through simulations using Chord as the underlying algorithm determined that two and three layer networks are sufficiently capable of reducing routing latencies without introducing a considerable amount of overhead [172].

In [173], the authors address certain best practices related to the use of super-peers in hierarchical networks. Using a two-layer P2P network, the network is organised into star-shaped constituencies with each peer connected to one super-peer. The super-peers then connect with each other to allow for full-network connectivity, as shown in Fig. 3.12.

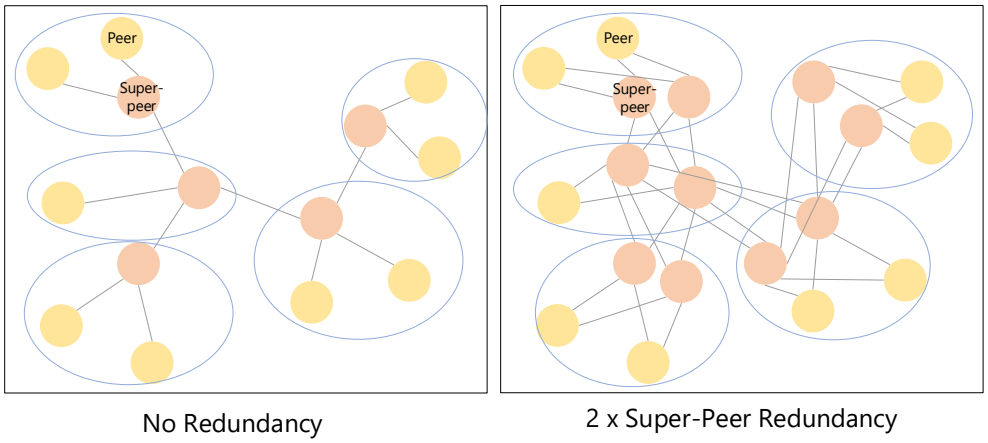

Figure 3.12: An example of a super-peer network with no redundancy and 2 super-peers redundancy as adapted from [173].

The authors experiment using this topology to determine the effect of increasing cluster sizes, the use of redundant super-peers, and the maximisation of the number of neighbours that a super-peer has, and the TTL values to be used. First, it was found that increasing the cluster size decreases the aggregate load at the cost of increasing the individual load. Second, the use of two super-peers (super-peer redundancy) in the place of each one does not impact the aggregate load, but notably reduces the individual loads. Third, increasing the number of neighbours, referred to by the authors as maximizing outdegree, reduces individual loads if all peers do. Hence, the maximisation of outdegrees should be performed uniformly across the network. Lastly, the TTL should be minimised based on the expected path lengths (EPL). The authors provide a procedure for global design that includes an effective method by which the appropriate minimal TTL may be used to initialise the network. After initialisation, the authors recommend that the TTL be decreased as outdegree increases to maintain a stable value for the number of nodes that process each query [173].

In [174] a two-layer network is proposed. The upper layer uses Chord for the super-peers while the lower is composed of star-shaped connections with the leaf node peers, as shown in Fig. 3.13. Cost-based analysis is used to evaluate this design. It is shown that hierarchical DHTs perform better than their flat counterparts, and that there is a

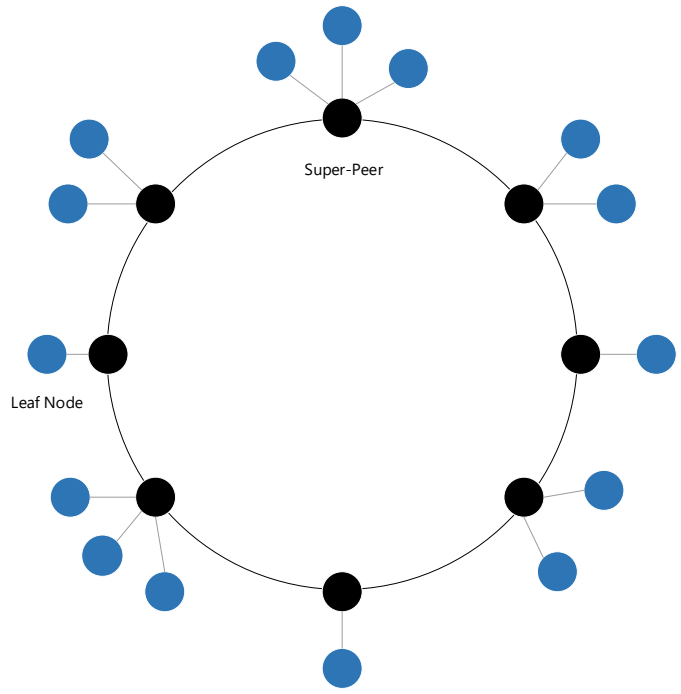

Figure 3.13: An example of a two-layer network in [174] with a Chord network upper layer and star-shaped connections with lower layer leaf nodes.

natural trade off between the costs for the highest loaded peer and the cost for the total network. This work is extended in [176] by performing a study on the various designs that may be used for connecting between the lower-layer peers. The aforementioned star-shaped design is found to be superior in performance to a fully-meshed structure, where all the peers within a star are interconnected and to one which uses a DHT to interconnect the intra-star peers. It's important to clarify that in all three cases there exist no direct links between the nodes of different stars. Thus, all communication must go through the respective super-peers [174, 176, 140].

With VeHA networks and their properties discussed, the next section will review the domain of P2P networks as systems.

3.1.4 P2P Networks as Systems

Another method of generating cross-layer integration in the enterprise can be found in the literature on networks as systems. Literature in this domain has been focused on bridging or merging together heterogeneous and homogeneous P2P networks to allow for expanded systems, inter-system content-sharing or inter-system traffic engineering. These topics will be addressed in the next sections in that order.

Inter-System Content Sharing Systems

Inter-system content sharing is when several systems cooperate together to share resources with each other [151]. The research present in this field, as related to structured networks, is focused on resolving issues in the inter-system routing of requests. The solutions presented primarily use co-located nodes [151].

In [177], inter-system routing is achieved by having a subset of nodes in an overlay network act as co-located nodes, or gateways, and then having a second group of nodes, known as the gateway pointers, keep track of these gateways. Nodes are designated gateways if they exist in two networks or are physically connected to a different DHT than its home network, as shown in Fig. 3.14. The proposed system exemplifies this principle by claiming that the proposed algorithms may potentially be used to connect 256-bit CAN, 160-bit Chord, and 256-bit Chord DHTs. To traverse across these differing networks, queries go from the origin node to a gateway pointer, then to the gateway, and finally to the external network. The selection of gateway pointers occurs without the need for any negotiations. To elaborate, when locating a gateway pointer, a common identifier is mapped to a keyspace id and the query is routed. If the keyspace that is being sought after is hosted locally, the node realises that it is a gateway pointer. The common identifier hashed once is the first gateway pointer, while the m^{th} hashing is the m^{th} gateway pointer. In this way, the process of selecting gateway pointers requires the modification of all nodes. The evaluation of this protocol focused on determining the setup overhead and operational efficiency. It is unclear if the results on the setup overhead were determined through experiments or simulation, but those of the operational efficiency were based on neither one. According to [151], the system was not evaluated using simulations or experimentation [177, 151].

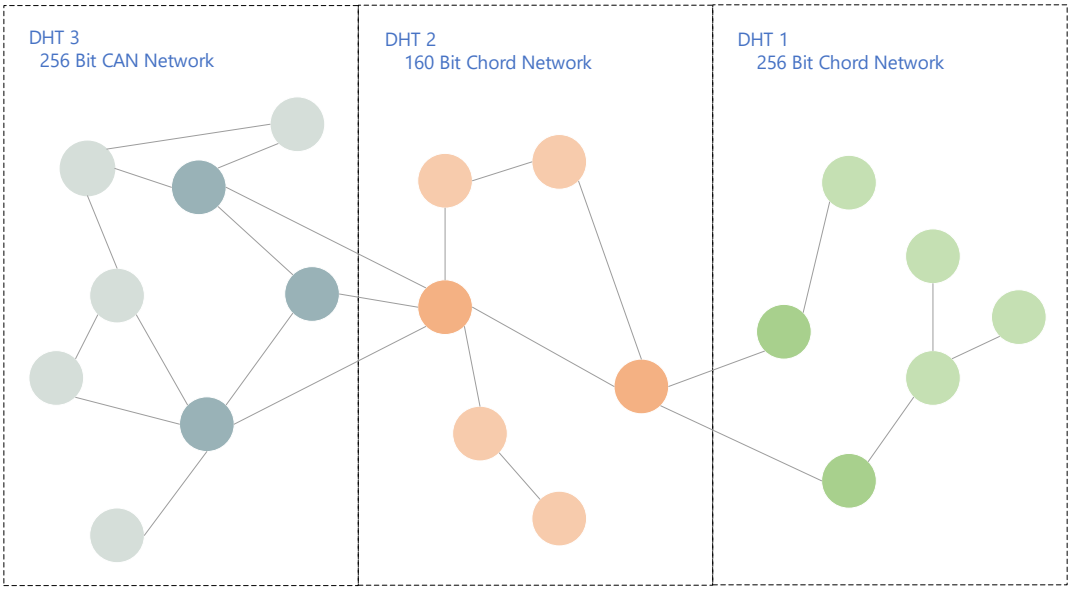

Figure 3.14: An example of three bridged DHTs for inter-system routing as adapted from [177].

In [178], Babelchord, a protocol for the bridging together of different Chord networks, is presented. Babelchord relies on the use of Synapse nodes, which are co-located nodes that belong to multiple rings, or floors, at once. To join a floor, a peer makes an offer based on the resources that it is able to add to the floor. The more resources that the peer has and the more applicable that they are to the floor the greater the peer's chance that it be permitted to join the floor. To participate in multi-flooring, nodes require extensions to their hashing methods and routing table. For example, in the case of a node that participates in multiple floors, its routing table must be extended in order to take ensure that the node is aware of its successor and predecessor on the ring for each floor. When a query passes by this node, the lookup is forwarded to all of the floors that the node is a part of. This requires that the node also know the different hash functions of the floors that it is a member of. To limit unnecessary lookups, previously processed requests are discarded. Also, a TTL value is assigned to each query and decremented every time it crosses a floor boundary. To optimise lookups, Babelchord allows nodes to keep lists of the different peers and floors that have resulted in successful lookups, referred to as hot peers and hot floors. Optimisation may then be achieved by inviting a hot peer to join one of the node's floors, or the node may join a hot floor, or, lastly, a new floor may be created. The Babelchord protocol is evaluated using a custom simulator written in Python. Simulations vary the network size, number of floors, and number of floors that each peer belongs to. Results demonstrate the protocol's scalability and show

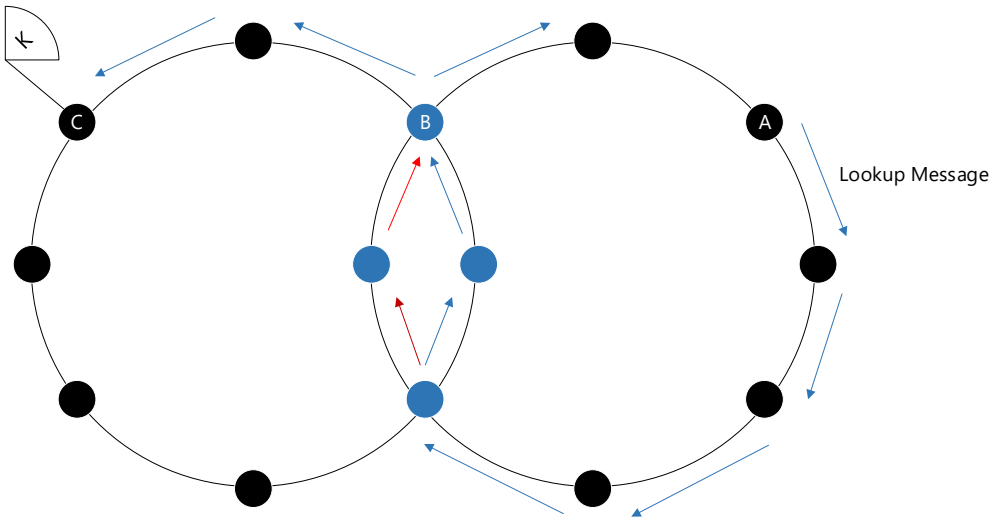

Figure 3.15: An example of Babelchord lookup routing in two floors as adapted from [178]. Node A queries for resource K on node C. The lookup from node A reaches node B twice. The query following the red path is terminated at node B as it has already been processed by node B via the blue path.

that only a comparatively small number of synapse nodes is required between floors to achieve over 50% exhaustive lookups [178].

In [179], the same authors from [178] present a second unstructured protocol termed Synapse, based on the concepts of Babelchord's synapse nodes. Synapse is developed for content sharing between heterogeneous overlay networks. It is available as a "white box" and "black box" protocol. The former is designated for use with protocols and software clients that are open to modification. The white box approach therefore aims to integrate additional parameters to allow for intra-network routing. Every node therefore needs to be aware of the changes made to their network protocols. Black box routing on the other hand is designed for use with networks that utilise proprietary protocols or protocols that are not subject to changes. Consequently, the black box model uses a Synapse control network instead to maintain the parameters that networks with immutable protocols are incapable of incorporating freely [179].

Effectively, the white box Synapse protocol is the same protocol as Babelchord with extensions to allow for message passing between networks of different DHTs. Thus, every query received by a co-located synapse gives the node the option to start a lookup process in all its other connected overlay networks. The non-hashed key and node addresses should therefore be available to synapse nodes so that they may be hashed using the appropriate hash function before being routed across the adjacent overlay. A

key can be stored on several peers to increase robustness and availability in more than one overlay network. A Maximum-Replication-Rate (MRR) value is therefore used to limit the number of resource replicas. The "game over" strategy for limiting queries and preventing routing loops is inherited directly from Babelchord. This implies the use of TTL values to drop drop over-extended or duplicate messages and message tags to drop previously processed lookups, respectively. The parameters for correct inter-overlay key and node address hashing, key replication, and the game over strategy need to be integrated into the protocols of the underlying overlays [179].

The black box protocol assumes a network of "blind" peers running the unmodified protocol of its overlay and synapses connecting the overlay to one or more other networks. The synapses communicate and share routing information with each other using the control network. This network is composed of a set of three DHTs implementing a Key table, a Replication table, and a Cache table. The Key table stores the non-hashed keys of the bridged overlays. The Replication table stores the keys' MRR values. The Cache table implements the replication of key retrieval requests, caches responses, and "control[s] the flooding of foreign networks" [179].

The white box Synapse protocol is evaluated using a lightweight prototype, JSynapse[1], and an open source prototype, open-Synapse[2]. The latter is based on the open source implementation of Chord, open-Chord[3]. While both prototypes are deployed on the Grid5000 platform, a test bed made of thousands of interconnected machines across several sites in France, JSynapse undergoes real deployments as well. The evaluation of the white box approach showed that it was resilient in the face of churn, and, similar to Babelchord, that it was capable of near-exhaustive searches using a low number of Synapses, but only if a TTL parameter is included to limit the communication overhead. The black box approach was not evaluated [179].

In [180] and [181], modifications are proposed for Synapse's black box method. The control network is removed from the protocol and replaced by a Synapse discovery mechanism to allow the synapses to detect each other's presence in networks. Four discovery mechanisms are proposed: message embedding, active notifications, peer exchange, and aggressive discovery. In message embedding, a node adds a list of the networks that it is a part of to every message that it communicates. This, requires modifications to the protocols of the underlying overlays. The second method, active notifications, has a synapse node proactively respond to the source of any message transiting through it with a *SYNAPSE_OFFER* message, thus makings its presence known. The third method, peer exchange, is identical to the concept of peer bootstrapping. Finally, aggressive discovery is a generic name given to the possibility that the Synapse protocol may exploit unspecified attributes of underlying protocols for the purpose of discovery. The discovery mechanisms afford synapses the ability to maintain pointers to each other. This information is stored in a Direct Overlay Table (DOT) and is used to route messages across overlays by having synapses directly contact each other in their respective over-

[1] http://www-sop.inria.fr/teams/lognet/synapse-net2012/

[2] No longer available on the INRIA Logical Networks team webpage.

[3] http://open-chord.sourceforge.net

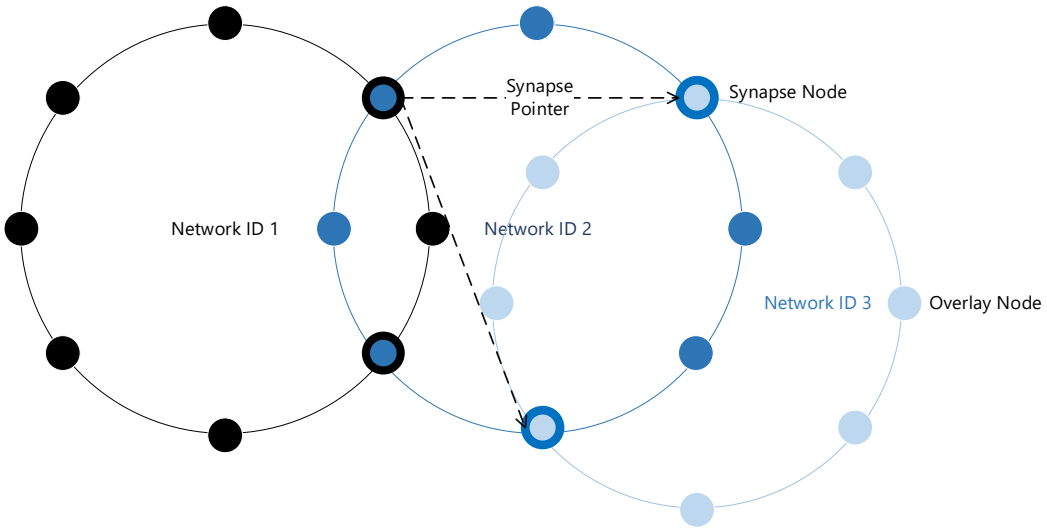

Figure 3.16: An example demonstrating a Synapse overlay.

lays, as shown in Fig. 3.16. It's also worth noting that synapses also maintain a separate Message Routing Table (MRT) that is used to keep track of the ongoing messages.

The proposed protocol is evaluated using Oversim (an overlay simulator built atop Omnet++) and by deploying the Synapse protocol on the Grid'5000 test bed. The simulation tests the protocol in a churn-less and high churn environment running 2000 nodes in an equal number of Chord and Kademlia networks. Two topologies are used, FIT and RAT. The FIT topology represents a fully-interconnected overlay with a path existing to any other overlay through gateway nodes. The RAT topology is composed of completely random assignments of overlays to gateways. Results show that performance is not highly correlated with the topology used. Therefore, a high knowledge of the underlying overlay systems is not required when building a Synapse network. This assumption holds with a system of 200 overlays as results show exhaustive searches. However, more synapses are required for exhaustiveness if synapse nodes have low connectivity in the network [180].

The Grid'5000 experiments use a system of 1000 nodes distributed across 10 Chord and 10 Kademlia networks that are interconnected with synapses. The system is found to be resilient to churn when the mean lifetime of nodes is above 900 seconds. Exhaustive searches are also possible in the network given the same average lifetime of nodes. To counteract shorter lifetimes, a higher degree of connectivity between synapses is required. There exists a cut-off TTL at which search exhaustiveness plateaus and no longer increases significantly with increasing TTL. Similarly, for this network size, assigning 20% of the peers to the role of Synapse nodes is sufficient to achieve exhaustiveness [180].

In [182] and [183], the works of [180] and [181] are extended and renamed as the Overlay Gateway Protocol (OGP). An important differentiator between OGP and previous works is that OGP puts nodes of a single overlay into a single group instead of a single DHT. Therefore, OGP is able to bridge together structured and unstructured networks.

The OGP protocol designates nodes as either Blind peers, Full OGP peers (FOGP), or Lightweight OGP peers (LOGP). Blind peers are completely unaware of the OGP protocol affording OGP backward compatibility. FOGP peers are the same synapse nodes that can be used to generate the Synapse overlays described in [180] and [181]. LOGP peers use inter-overlay routing for their own advantage but do not participate in the OGP overlay [182, 183].

FOGP and LOGP nodes and their overlays are topologically organised into a binary tree to allow for three forms of routing: OGP unicast, OGP multicast, and OGP broadcast. Unicast, multicast, and broadcast networking is traditionally known as the transmission of a message to a single node, a group of nodes, and all the nodes in a network, respectively. In OGP, these concepts take on new meaning in that they represent the communication of a message to a single overlay, a group of overlays, or all overlays [182, 183].

The OGP protocol is evaluated using the Grid'5000 network in two scenarios: 20 networks of 50 nodes each running Kademlia, Chord, and Gnutella networks, and 3 networks of 100 nodes each running BitTorrent, a Kademlia-based protocol, and Gnutella. The first experiment is used to evaluate the overall OGP framework and the second evaluates the framework's ability to support file-sharing capabilities. Results show that efficiency in routing and cooperation can be achieved using small numbers of FOGP nodes for each participating network, namely 5 peers per network. Routing costs are also found to be logarithmic. LOGP nodes are also capable of achieving the same routing efficiency as FOGP nodes while only minimally increasing FOGP traffic. This is explained by the observation that LOGP nodes only produce very little traffic. Thus, the OGP protocol is a scalable solution for inter-system content sharing [182, 183].

Merging P2P Systems

The second type, merging P2P systems, is when groups of nodes share the same goals. This means that the relevant nodes may be grouped together under the same location as one system eliminating the factor of competition between them [151].

In [153], the task of merging together distinct structured overlay networks that use the same protocol is observed. The study is conducted on two protocols, Chord and P-Grid, to determine the properties that affect the merger process.

For Chord, the ring-based structure is a large hindrance to a merger process. The operations of merging Chord overlays are completely disruptive until the merger is complete. This is because once the nodes of both networks are integrated into a larger one, the given probability that a node's predecessor changes with the addition of new nodes is $1 - e^{-\lambda_1}$, where $\lambda_1 = N_2/N_1$, and N_1 and N_2 are the number of nodes in networks 1 and 2, respectively. This means that the number of peers that will have their successor,

predecessor, or both, changed is equal to $N_1(1 - e^{-\lambda_1}) + N_2(1 - e^{-\lambda_2})$. These changes need to be communicated to the immediate and respective neighbours to allow all routing tables to be eventually corrected. Therefore, until the merger is complete, and the structure is once again stable, queries cannot be routed to the correct peer. The unique key value pairs of each network may also be inaccessible until they are redistributed amongst all the peers of the newly formed network. Another point to consider is that the replication of data across the merged networks may add a considerable strain on the nodes of the participating networks. Thus, the authors suggest that it is possible that certain nodes may be overwhelmed by the merger process and may subsequently leave the network. This hypothesis, however, requires an actual evaluation of a merger process which is yet to be done [153].

In comparison to Chord, the features of structural replication in P-Grid mean that routing processes are not disrupted during their mergers. Nodes with identical paths replicate their counterparts' content and merge to become redundant replicas of each other. If nodes in one network do not have a path route with the existing prefix in the other network then the path and nodes are retained without changes. Therefore, the functions of the network are unaffected by the merger. However, the resources of one network will not be visible to the other until the replication processes are complete and the lack of policies structuring or limiting the number of replicas prevents there being a deterministic way of knowing "the full replica subnetwork at each peer" [153], meaning that the replication process is probabilistic. This contrasts with ring-based mergers, which are precisely deterministic [153].

In [184] an algorithm is proposed for merging ring-based overlays. While the algorithm is claimed to be applicable to unidirectional ring-based structured overlay networks, the findings in [184] are based exclusively on the Chord protocol. The algorithm presented is capable of two types of mergers.

The first of these is concerned with merging two networks that were created through the partitioning of one network. Network partitioning is expected to cause a large number of nodes to be determined as failed nodes. The algorithm requires that these failed nodes be placed on a passive list that is used to track their state. Every node regularly checks its passive list to see if any of the members listed are once again reachable. Thus, any attempt by the partition to merge is detected and causes the merger algorithm to be executed [184].

The second type addresses the merging of two independently created networks. In this case, a living node from one of the independent networks must be inserted into a passive list of the other to trigger the algorithm [184].

Once the merger algorithm is triggered, the previously passive node is placed in a queue, *detqueue*. The queue is checked periodically and if it is found to contain nodes, the first of these nodes is selected for the merger process. A lookup with the node's ID is sent into the network. Once the lookup nears the destination, the predecessor and successor values of the nodes located there are updated. Recursive calls are then sent in both the clockwise and counter-clockwise directions to continue the merger process along the ring. It is claimed that this process, termed the "simple algorithm" is slow

and may fail under certain conditions. Therefore, a second gossip-based algorithm is also proposed. The gossip-based algorithm has the simple algorithm initiated at a random number of nodes. To reduce the messaging costs of this procedure, several strategies are implemented. First, instead of immediately initiating lookups for the living node at the random nodes, the living node is instead placed in its respective detqueue. Because these queues are checked periodically, the lookups are not triggered in unison. To decrease the resource cost of the algorithm even further, the number of random nodes selected is varied based on a fanout parameter that is decreased with every selection of a random node [184].

The proposed system is evaluated through simulations. However, the simulator employed is not stated in [184]. Simulations are carried out using 200 nodes with varying node counts ranging between 256 and 2048. A merger is considered successful if 95% of the nodes end up with correct successor pointers. The proposed algorithm is claimed to be largely successful because only 3 out of the 200 executions for the simple algorithm result in unsuccessful mergers while the gossip-based algorithm resulted in 0 unsuccessful mergers [184].

Inter-System Traffic Engineering Systems

The last type, inter-system traffic engineering systems, involves the use of P2P systems to optimise network routing performance. Different approaches have been suggested in [185] and [186]. These are described in the next few paragraphs.

In [185], the Synergy architecture for overlay inter-networking is proposed. Simply, this architecture uses co-located nodes to route messages over different overlays for performance gains. In more detail, each network participating in the inter-networking effort periodically assigns one of its nodes the role of an overlay agent. This agent is then used to aid its network in joining Synergy. The selection of a node as an agent is based on an estimation of how co-located it is. This estimation is reached, for example, by evaluating the number of overlays that the node is connected to, the number of overlays that its neighbours, past and present, are connected to, and the minimum and maximum latency experienced by this node within its home network. Constrained and/or loaded nodes are disqualified from the selection process. For redundancy, every agent is given a backup. Agents across all of the participating networks use a bootstrapping mechanism to form a second overlay, the Synergy network, which then allows for cross-overlay inter-networking. The Synergy network is used primarily for cooperative forwarding, where traffic is permitted to flow outside of the network using Synergy routes. By providing a larger pool of nodes, network routes may be optimised to avoid network hotspots, reduce routing latencies, or enhance the overall stability of the network. However, to avoid overloading the Synergy routes, three factors are imposed on the network. First, each host is limited to a specific number of flows. Second, only one flow is permitted to choose its routes at a time. Lastly, flows determine if hosts are to be used as parts of their paths by utilising a prioritisation algorithm that uses an exponentially weighted moving average of the number of flows that have used the host within a given time period. Also, the Synergy network uses link state for the computation of routing paths. The proposed

architecture was evaluated experimentally. Results showed that the use of cooperative forwarding improved overlay connections' latencies, throughput, and packet loss [185].

Lastly, [186] suggests the sharing of resources between overlays to improve specific network performance characteristics. The first action required by the protocol is for all of the peers to join a mesh-based overlay. The peers need to be bootstrapped to allow them to locate and construct logical links with a number of peers. The Mesh-Based Overlay Manager (MBOM) is then used to have the generated mesh maintain fidelity to the underlying physical topology. For topology matching, the MBOM uses the Location-aware Topology Mechanism (LTM). However, the MBOM also uses flooding-based detection to discover new neighbours and timestamped updates to determine and eliminate low-performance links. The Single Overlay Manager (SOM) manages the selection of the paths of least delay for source-to-destination communication flows. Timestamps are used to determine the most useful paths. The Inter-Overlay Optimisation Manager (IOOM) is responsible for the management of backup streaming path sets and active streaming path sets. The former involves the selection of backup paths for nodes using a reverse tracing algorithm. The algorithm notes when the number of paths in a node's backup path set drops below a certain threshold. At this point, the affected node communicates a probing message to a number of its neighbours with an empty array. Each receiver injects their ID and timestamp into the array. If a receiver finds that the total calculated delay is larger than the previously known least delay, then it does not forward the message. Likewise, if the receiver was the source of the message, then the message is not forwarded. Otherwise, the message is forwarded again by the receiver to the same number of random neighbours as was done by the original message source. This way, the original source may be able to calculate and construct the best possible paths. The management of the set of active paths involves the maintenance of the currently active path, the removal of failed links, and the addition of new active paths from the backup set. The key node manager performs admission control functions to avoid path over-subscription. Finally, the buffer manager ensures that the streaming application has valid data from multiple sources to maintain continuous playback [186].

3.2 Discussion: Selecting a Protocol

Based on the reviewed literature, of the various types of P2P networks possible the one that is decidedly the most suitable for the task at hand is that of P2P networks as cooperative systems. This is because this class of networks bridges or merges together existing P2P networks to allow for expanded systems of networks, inter-system content-sharing, and inter-system traffic engineering [151]. Using the concepts of this domain, devices may therefore be capable of organizing themselves into systems of cooperative nodes. Each layer of the enterprise may be comprised of one or more complete and self-contained overlay networks. Using the concepts of cooperative systems, new systems may be dynamically formed to bridge together the nodes from the different overlays of the enterprise to offer new services or to execute new functions and processes. Consequently, by allowing for the dynamic reconfiguration of systems, this subdomain may therefore

be used to facilitate or restrict message and content transfer between overlays and enterprise layers dynamically, and possibly autonomously, in accordance with the enterprise's changing requirements. Such a subdomain therefore allows the proposed resulting manufacturing infrastructure to establish pervasive enterprise-wide communication channels that are resilient to failure without violating or compromising the binding constraints of the industrial architecture.

Establishing reliable routing mechanisms for a distributed system operating using the principles of P2P networks as cooperative systems directly translates to the need for a reliable inter-system routing policy. Unfortunately, as is apparent from the previous sections, work in this domain is scant. It is however possible to derive the fact that, for enhanced reliability, solutions have converged towards the use of either co-located nodes or gateways. The difference between the two strategies, as defined in [187], lies in the fact that co-located nodes participate in the routing process, while gateways only maintain pointers towards specific nodes in different overlays. Works that representatively exemplify these principles have been presented in Section 3.1.4.

Based on the results of previous studies, it appears that regardless of whether an exclusively gateway-based or co-located-nodes-based method is used, the basic components of an optimal inter-system routing protocol are nodes, gateways, and gateway pointers. The first step in developing an inter-system routing protocol should therefore be instilling the modifications necessary to allow for the inclusion of these elements.

To exemplify these principles, the next section will describe their usage in developing a functional cooperative systems P2P protocol. However, due to the limited scope of the study and the large number of existing and mature P2P solutions based on proven methods, an existing single-system solution, Chimera, is used as a starting point for the development process. Chimera is selected because it is a hybrid implementation of two well established P2P protocols, Pastry and Tapestry. Its hybrid nature means that it has inherited a favourable number of properties and constituent elements that are also present in several other largely popular P2P protocols. This lends credibility to the applicability of the methods of Section 3.3 for the conversion of other existing P2P protocols into cooperative systems.

3.3 Developing a Cooperative P2P Network

This section demonstrates the conversion of a traditional P2P protocol, Chimera, into a cooperative P2P routing protocol. The developed mechanisms are explained and the results from the experimentally tested implementation are presented and discussed.

3.3.1 Design

The design process is described in two parts. The first presents the Chimera protocol in its original form. The second explains the conversion process that transforms Chimera into an inter-system routing protocol.

Base Protocol & Modifications

This subsection will explain the main elements of Chimera to establish its baseline behaviour. These elements may be summarised as bootstrap nodes, prefix-based routing, fault detection through the use of heartbeat messages, neighbour sets, and leaf sets. Due to a lack of publications on Chimera, the descriptions provided are inferred from a library implementation in C that was developed at the University of California - Santa Barbara[4].

The definition of each of the aforementioned elements of Chimera is as follows. First, a bootstrap node is a member of a P2P network that supplies newly joining nodes with the initial configurations required for them to successfully join the network. Next, prefix-based routing is a form of message routing that uses the unique identifiers or keys of nodes in routing. In prefix-based routing, the next hop selected for a message is the one that matches the prefix of the destination with a digit extra than the current hop. As for fault detection through pings, or heartbeat messages, this mechanism relies on the successful acknowledgement of pings between nodes to surmise that other peers are functional. A node's neighbour set consists of information on the peers that are closest to it in terms of proximity. This set is not typically used for routing, but is meant to be used as a source of locality information. Finally, the leaf set of a peer, L, is composed of the nodes with $|L/2|$ numerically larger and $|L/2|$ numerically smaller node IDs as compared to the peer's own node ID. Unlike the neighbour set, the leaf set is used for message routing [143, 160].

Together, the aforementioned mechanisms operate as follows. Initially, Chimera must be supplied with the details of a bootstrap node or be initiated as the bootstrap node. If it is initiated as the bootstrap node, then it simply waits for join requests. If it is supplied with the host name, port and key of a bootstrap node, then, as shown in Fig. 3.17, the joining node, A, transmits a join request containing its host name, port, and key to the bootstrap node, B, and waits indefinitely for a response. If node B accepts the join request, node B sends node A its leaf set, which is processed by Node A and subsequently used to send an update message to each of the members of the leaf set announcing its arrival.

Once the join process is complete, the network is then maintained through the use of heartbeat messages and what is termed as piggyback messages. To elaborate, every preset length of time a node is expected to ping every other node in its leaf set. If the pinged node responds with an acknowledgement within a pre-assigned grace period, the node is acknowledged as a functional one. If, however, the node does not respond in time and its success average is found to be below an acceptable threshold, the node is pruned out of the routing table. As for the piggyback messages, these are communicated after every third ping. This message type contains the entire leaf set of the node, and is communicated to every member of its leaf set in order to disseminate routing table updates.

Further details of the Chimera implementation relate to its external and internal host

[4]Available: http://current.cs.ucsb.edu/projects/chimera/

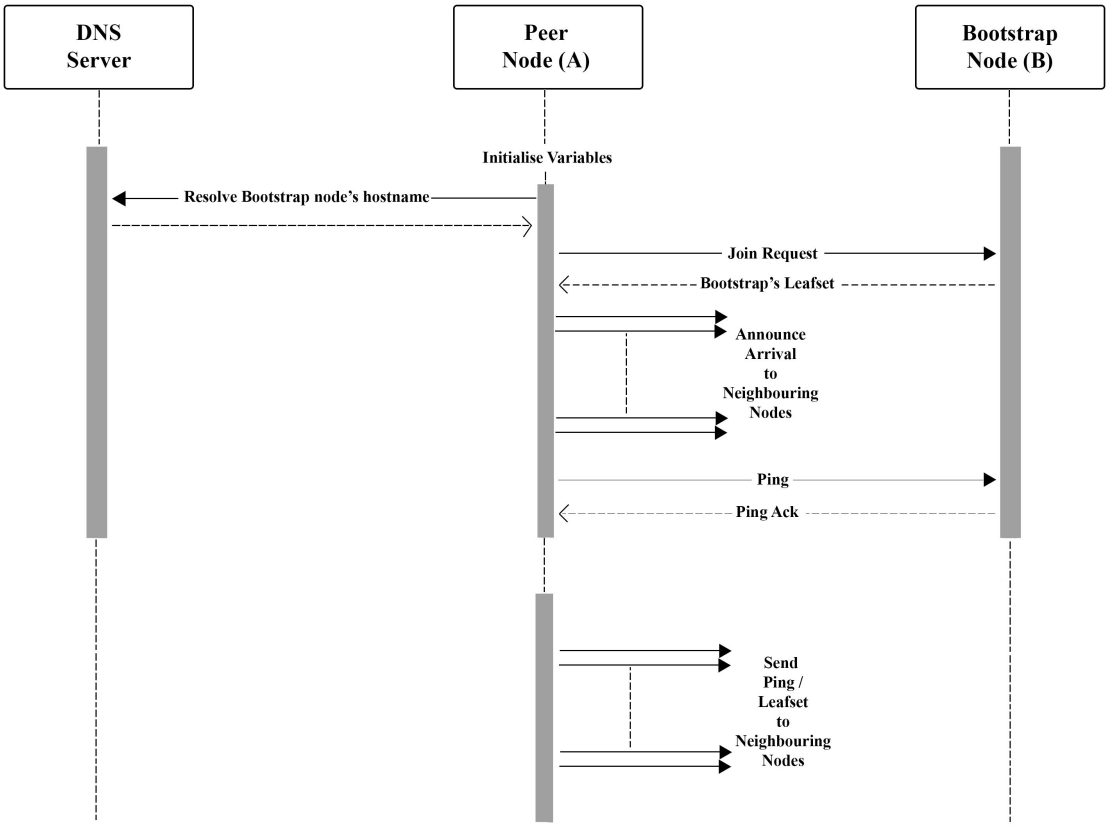

Figure 3.17: The operational mechanisms of the vanilla Chimera P2P protocol.

lookup algorithms. For external host lookups, Chimera is dependent on DNS lookups for host resolution. The resolved IPs of hosts are not stored and lookups are performed before every transmission. The remaining information related to peers, such as host names, keys, and ports, are stored using the Red-Black binary tree library. As such, these binary trees are used for subsequent lookups of information on hosts, as well as for other implementation-specific behaviour relevant to the proper functioning of the program itself.

All of the aforementioned communication occurs solely through the use of UDP packets. The structure given to the UDP packets is shown in Fig. 3.18. Of the header elements listed, the seqNum and source key fields are not used by the original authors of Chimera. That is, the message header struct includes these fields, however, they are not included or used in any transmitted or received packets.

Some of the limitations of Chimera include the fact that support was not extended for TCP communication. Furthermore, aside from the provision of UDP-based message transferring mechanisms, no functions were defined for resource lookups, file transfers, or any other cooperative behaviour typical of P2P networks.

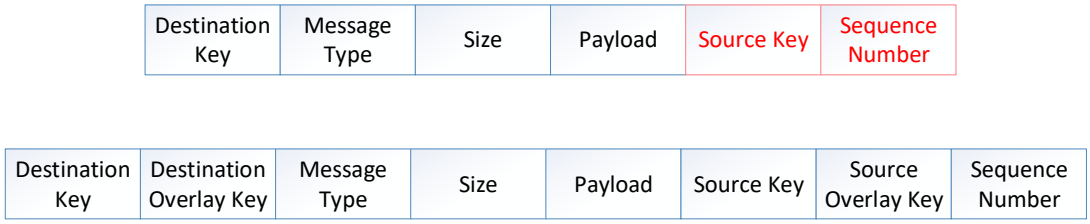

Figure 3.18: The structure of a UDP Packet in the original Chimera implementation (above), and post-modifications (below).

Modifications

Since the purpose of modifying Chimera is to instil the necessary measures to allow for inter-system routing, the first requirement is a method by which overlays may be differentiated from one another. To do so, the protocol mirrors the same identification method used to differentiate nodes from each other. That is, as each node is given a unique key, each overlay is also assigned a key. Although it is a simple change, this modification echoes throughout the routing, messaging, and higher level functions and layers.

Starting with the messages themselves, a message must be clearly labelled with its destination overlay. Consequently the message header is modified to include this key as well. Furthermore, the source key and sequence number header fields are enabled and a source overlay key field is added. These three fields are included for reasons that will be clear later in this subsection. The resulting message header is shown in Fig. 3.18.

As for the routing process, naturally, the first step is to define a third node type that allows a node to operate as a gateway. This role is denoted as the 'co-located' node type and is typically considered to be fixed. This is because, in an industrial setting, nodes that would be responsible for cross-layer communication will need to be granted the proper access rights to do so. This means that such gateways might need several types of communication interfaces, additional wiring, multi-homing, or the configuration of firewalls and other devices to allow for the traversal of gateway-specific traffic. Since only a number of devices may be afforded the configurations or hardware necessary to operate under these constraints, the P2P protocol must therefore be able to accommodate the use of these nodes as fixed gateways.

Currently, the role of a co-located node is designed to be occupied exclusively by bootstrap nodes. That is, a co-located node is always a bootstrap node, but a bootstrap node is not always a co-located node. Since all nodes must know the bootstrap node to join an overlay and Chimera operates by using fixed bootstrap nodes, by doubling the role of a bootstrap node as a gateway node, all of the members of each overlay become de facto gateway pointers. This reduces the complexity of the protocol by eliminating the need for the selection of gateway pointers and gateway discovery processes.

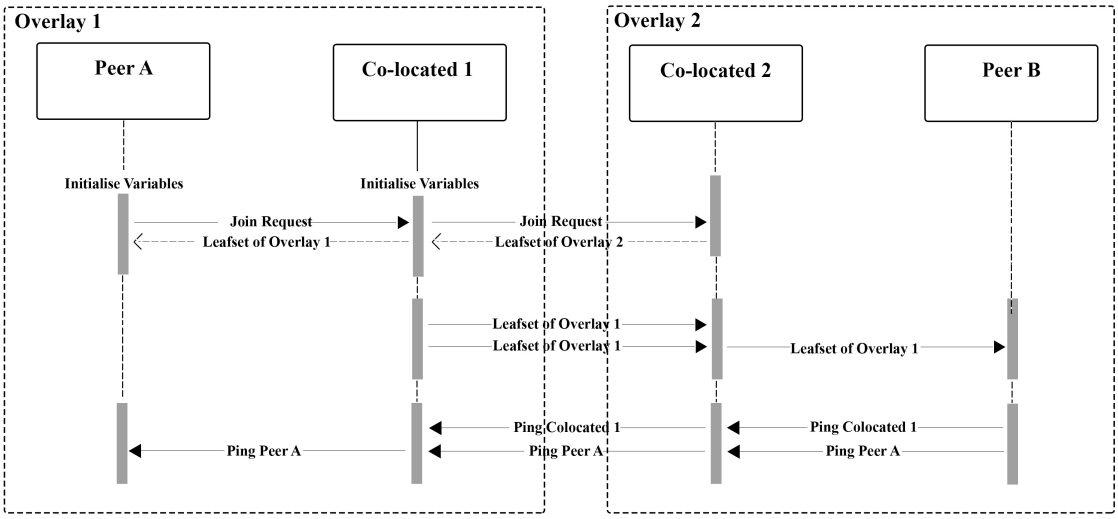

Figure 3.19: The operational mechanisms of Chimera after modifications.

With all of the aforementioned modifications in mind, the message routing process is, naturally, modified as well. First, due to the geographic proximity of peers, prefix-based routing is abandoned. Instead, if a message is destined to a node in the same overlay as the sender, then the message is sent directly towards the destination. However, if the message is sent to a node in a different overlay, then, as shown in Fig. 3.19, the message is routed via the co-located node with the destination key and destination overlay key clearly labelled. If the bootstrap node contains a listing of a co-located node in the destination overlay, which in this case would be a node that has joined its overlay from the destination overlay or vice versa, then the bootstrap node forwards the message to the co-located node. In turn, the co-located node then forwards the messages towards its final destination. If no co-located node in the destination overlay is available, then the bootstrap node transmits the message to a co-located node from a randomly selected overlay in the hopes that it may have a path to the destination overlay. If the bootstrap node is not aware of any active co-located nodes, then the message is dropped.

As previously mentioned, the message header is modified to actively use the message layer sequence number, source key, and source overlay key. The source key and overlay key are used for two purposes. The first is to instil a loop-breaking mechanism which was not originally present in Chimera. That is, the next hop of a message is prevented from being the previous hop to avoid routing loops. The second purpose is to simplify the experimental analysis process. By using the source key, overlay key, and sequence number, messages can be traced back from the destination to the original source permitting the collection of metrics such as the number of hops travelled per message. Beyond the experimental value of including these three fields, they may also serve to enhance the security of the protocol. This was their intended purpose in the original implementa-

tion of Chimera, but, like vanilla Chimera, they are currently still not used for security purposes.

Further routing-related changes involve the modification of the purpose of leaf sets and neighbour sets. As prefix-based routing is not used, leaf sets are therefore instead used as overlay sets. That is, each leaf set maintained by a node corresponds to an overlay. Consequently, as shown in Fig. 3.19, the pinging and piggyback messages have also been modified to accommodate for this change. The pinging function now operates by having a node select a leaf set at random before pinging all of the nodes in that overlay. As for the piggyback messages, like the pinging function, the destination leaf set of the piggyback message is chosen at random. However, the contents of the piggyback message always contains information on the nodes in the transmitting node's own overlay. Lastly, the second type of sets, neighbour sets, are not currently used. However, because they may be useful for the execution of other functions, such as load distribution and data replication, they are kept as vestiges within the current implementation for future use.

The final modifications done to Chimera are related to increasing its efficiency and flexibility. For the former, Chimera's dependence on DNS-based host name resolution is eliminated. Instead Chimera directly stores and shares information on host IPs through piggyback messages. For the latter, further flexibility is achieved through two measures. The first is the modification of all the functions responsible for network communication to allow both 32 bit and 64 bit platforms to execute Chimera. This is to allow the system to use computing resources from both ends of the spectrum of available devices. The second modification is the removal of Chimera's dependence on the Red-Black Tree library. Instead, a generic hash table is instilled, which may be easily extended to use a Red-Black Tree, a NedTrie or any other digital searching mechanism that developers may desire.

The next section describes the experimental procedure and results used to evaluate the designed inter-system routing mechanisms.

3.3.2 Experimental Results

The protocol presented is designed to allow for the reliable inter-system routing of messages between clusters of nodes and this section describes tests of the protocol's abilities and limitations. These tests are performed while observing the degree of variation in the protocol's behaviour as the number of clusters and nodes per cluster are modified to establish a relationship between the two.

Results

The experiment is conducted on a Xen Project (TM) server hosting 48 Debian virtual machines (VMs). Two further VMs running Ubuntu are deployed separately on two 32-bit CubieTruck ARM boards that were also extended using the Xen platform. The physical network configuration therefore consists of two CubieTrucks connected to a single network port on the server using an unmanaged switch. The port is then bridged to 48 virtual interfaces, with each interface belonging to one of the 48 VMs.

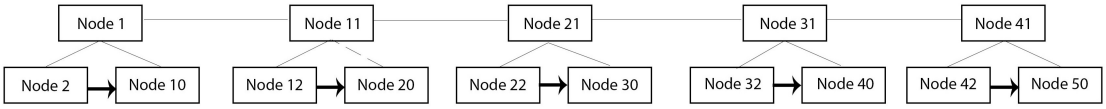

Figure 3.20: The structure of a 5 cluster network.

The number of clusters and nodes per cluster are varied to establish a relationship between the two variables. As such, a fixed number of nodes, 50 in all, are divided into 1, 2, 5, 10 and 25 clusters with each cluster consisting of 50, 25, 10, 5, and 2 nodes, respectively. Clusters are connected linearly via their bootstrap nodes as shown in Fig. 3.20.

The configuration and execution of these topologies is performed by a Bourne Again Shell (BASH) script running on the server's domain 0 (dom0)[5]. The script transfers the Chimera application with the necessary bootstrapping configuration files to all 50 nodes before each experiment. Once the transfers are complete, the same script executes all of the instances sequentially, with a gap of 2 seconds between each execution. The gap gives each instance enough time to initialise before it may begin receiving pre-configured messages, such as join requests, from any subsequently executed nodes. As a precaution, the Chimera instances are always run under the GNU Debugger (GDB). Finally, each experiment runs for an hour. During this time, each node collects and records information related to the behaviour of its network, routing, messaging, and application layers. Once the experiment timer expires, the same BASH script terminates Chimera on all 50 nodes, retrieves and archives all of the relevant logs, and prepares the node for the next experimental run.

The acquired logs are analysed in Matlab to evaluate the performance of the protocol. For each node, the total number of messages sent, acknowledged, received, and rerouted are calculated. Messages are also traced backwards, from destination to origin, to compute the number of hops travelled by every unique message transmitted during the experiment. The results are aggregated by cluster and summarised by means, medians, and 95^{th} percentiles. Since each of these forms of descriptive statistics may vary extensively from one node to the next, the maximum, minimum, and median values of each is also computed. All of the aforementioned results are shown in Tables 3.6, 3.7, 3.8, and 3.9. The case of having 1 cluster, where inter-system routing is not possible, serves the purpose of establishing the baseline behaviour of the protocol.

[5]Dom0 is the first domain started on boot by the Xen hypervisor (c.f. https://wiki.xen.org/wiki/Dom0)

	Messages Sent								
	Mean			Median			95^{th} Percentile		
# of Clusters	Min	Med	Max	Min	Med	Max	Min	Med	Max
1	2180	2180	2208	2166	2185	2174	2295	2312	2833
2	1848	1859	1876	1669	1683	1820	2349	2465	3033
5	1438	1642	2226	1222	1424	1844	3241	4243	5440
10	1368	1396	1777	1304	1368	1466	2047	2557	5294
25	621	698	750	502	724	812	1018	1109	1153

Table 3.6: The number of messages sent per node.

	Acknowledgements Received								
	Mean			Median			95^{th} Percentile		
# of Clusters	Min	Med	Max	Min	Med	Max	Min	Med	Max
1	2080	2121	2126	2079	2097	2129	2202	2207	2688
2	1669	1701	1737	1477	1556	1642	2068	2367	2643
5	1378	1583	2182	1165	1384	1802	3108	4201	5313
10	1344	1372	1740	1274	1345	1433	2039	2488	5213
25	611	692	740	493	723	809	1005	1092	1133

Table 3.7: The number of acknowledgements received per node.

	Acknowledgements Sent								
	Mean			Median			95^{th} Percentile		
# of Clusters	Min	Med	Max	Min	Med	Max	Min	Med	Max
1	2093	2126	2132	2100	2137	2138	2276	2290	2349
2	1673	1707	1741	755	995	1042	853	1105	1162
5	1384	1588	2186	392	436	462	9897	11757	16699
10	1348	1374	1433	274	286	341	5775	6386	7186
25	613	692	741	370	378	474	1292	1462	1515

Table 3.8: The number of acknowledgements sent per node.

	Number of Hops								
	Mean			Median			95^{th} Percentile		
# of Clusters	Min	Med	Max	Min	Med	Max	Min	Med	Max
1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1
5	1	1	2	1	1	2	2	2	2
10	1	2	2	1	2	2	2	2	3
25	2	2	2	2	2	2	2	2	2

Table 3.9: The number of hops per message.

From Tables 3.6-3.9, it is apparent that the number of messages transmitted and received follows a crude bell-curve shape. The maximum number is achieved at the mid range, where the number of clusters is half the number of nodes per cluster. Although the case where the nodes per cluster is half the number of clusters, trails closely behind. These higher recorded numbers of messages are due to two properties related to these clusters' configurations. The first is that there are enough nodes per cluster to allow for a healthy number of intra-routed messages. Second, the number of clusters is small enough to allow for inter-cluster discovery within one hour. This is apparent from Table 3.9, where the 5 cluster and 10 cluster cases are the only ones with means, medians, and 95^{th} percentiles reflecting the dominance of multi-hopped messages in their experiment runs.

Contrasting with these results is the highly granular case of having 25 clusters of 2 nodes each. Here, the number of transmissions are markedly lower than has been observed in the other experiments. This is because each leaf set is so small in size that a ping run on a leaf set results in only a few message transmissions before the source node rests and waits for the initiation of the next ping run. Although this means that the overhead of pings on the network is lower, it also means that in a fixed set of nodes, the discovery of faulty peers, and the subsequent pruning of their information, is also slow. Furthermore, the inter-cluster discovery process also suffers with this large number of small clusters. As shown in Table 3.9, the value at 25 clusters is consistently 2 hops, meaning that, within the time span of an hour, it was rare for a cluster to be able to discover clusters beyond its immediate neighbours. This is also a reason for the lower total number of transmissions observed for this configuration as messages did not have to travel far to reach their final destinations.

Although the protocol performed formidably, none of the experiments achieved a perfect record with respect to all sent messages being acknowledged. This is due to two possible explanations. The first is that once the hour-long experiment is complete, the termination of all instances is not instantaneous. This leads to the loss of acknowledgements, as they are either not sent or not received by a terminated application. Furthermore, at certain times multiple nodes try to communicate with the same one. This led to collisions and the loss of messages or acknowledgements. This is especially true for the cases of 2, 5, and 10 clusters, where co-located nodes were consistently under exceptionally heavy loads.

3.3.3 Discussion

Results from experiments, described in Section 3.3.2, indicate that having the number of nodes be double the number of clusters, or vice versa, is the best configuration for the protocol. Unfortunately, even with this configuration, the discovery process is considerably slow. The procedure of having to send hefty piggyback messages containing information on all of the nodes of a leaf set requires replacement with a quicker and more robust solution with less overhead.

The inter-system pings, although unnecessary for the maintenance of whole networks, were introduced to simulate the inter-system routing of messages. The protocol and

implementation is agile and stable enough to consistently allow messages to reach their destinations without fail and with the shortest path possible. Although this would allow designers to set conservative TTLs to messages, it comes at the cost of bootstrap nodes being considerably loaded with the task of re-routing messages throughout its operational lifetime. This brings to question whether such nodes would be able to participate in any other tasks required of a node beyond routing. A further consideration is that, due to these high loads, implementing star-shaped clusters is not a realistic option. Redundancy and load balancing should be considered to offset inevitable failures that would result from the heavy routing loads that co-located nodes must handle. Finally, another desirable features is a mechanism that allows for a mesh configuration between the co-located nodes. This is preferred over to the linear routing currently in place in the hopes of reducing routing loads and message latencies.

Finally, limitations of this experiment are discussed. The gaps between initiation and termination naturally mean that all experiments did not run for exactly an hour. The effect of this limitation has been noted in the Section 3.3.2. Another limitation is that the experiment was only run for an hour each time. In the future, it would be interesting to observe the steady state behaviour of the nodes in the 2 or 25 cluster network after several hours, when wider knowledge of further clusters is established and inter-cluster routing plays a much more dominant role.

3.4 Service Discovery with mDNS & DNS-SD

The evaluation results of the developed protocol in the previous section highlighted two conclusions. The first is that the optimal network structure should have the number of overlays be double the number of nodes per overlay, or vice versa. The second observation showed that the discovery mechanism of the protocol, which is dependent on unicast transmissions of leaf sets, is detrimental to the performance of the network. These messages both added traffic to an overloaded network, especially affecting the co-located nodes, and were slow to deliver information necessary for the discovery of peers and overlays. For this reason, the networking protocol is converted into a service, the unicast-based discovery feature of the protocol is disabled, and service discovery is outsourced to an independent module. A management service is also developed to coordinate between the networking and discovery services.

Another modification is the extension of the protocol implementation with IPv6 capabilities in accordance with the principles of contemporary designs.

The next subsections describe the discovery and management services and their interactions with the networking service. The implementation and experimental evaluation of the overall system is also presented.

3.4.1 The Discovery Service

This subsection is concerned with the selection and implementation of a robust and efficient discovery mechanism for the overlay system. It does not aim to provide an

exhaustive list of all existing methods for discovery. Rather, the discussion is limited to mechanisms found in other P2P networks as cooperative systems and one from the domain of zero configuration networking.

Although the field of cooperative P2P systems is relatively new, it has several implementations such as [188] and [187] that provide a number of possible methods for discovery. These two implementations are described in the next few paragraphs.

In [188], a framework designed to enable the cooperation of heterogeneous P2P networks is presented. Node discovery within networks is left to the protocols of the respective overlays. Whereas the discovery of co-located nodes is handled by the mechanisms of [188], which involve the use of flooding or a secondary overlay network called the Internet Indirection Infrastructure (i3). The former, flooding, is not an acceptable option because it would only create the same network conditions and strain as occurred in the first iteration of the modified Chimera protocol. The i3 overlay based method, based on the work of [189], depends on the use of a network of servers, called the i3 network, that behave as discovery and forwarding servers. For example, when a node becomes a cooperative node, it registers itself with an i3 server using a 'trigger' message containing a service identifier. Previously registered nodes periodically check in with the i3 infrastructure by pushing a packet type message containing a service identifier and their network address. The i3 servers forward these messages to other nodes registered with the same or similar identifiers. Once the new node receives this informational packet, it de-registers its trigger from the i3 overlay. Applied to the distributed manufacturing infrastructure, this method would require the creation of a secondary infrastructure to support the discovery of nodes. The daunting complexity of this task causes us to look elsewhere for suitable solutions.

In [187], local discovery is managed by the respective protocols of the member overlays. However, as previously mentioned in Section 3.1.4, the discovery of co-located nodes, termed synapse nodes in [187], is done using message embedding, active notifications, peer exchange, and aggressive discovery mechanisms. For message embedding, a node includes all of the overlays that it is connected to in outgoing messages. Any synapse node that is part of the message path decodes this information and updates its tables. For active notifications, a synapse node, becoming aware of a transiting message, pro-actively notifies the sender of its connected overlays. Peer exchange involves the iterative transmission of discovery-relevant information. Finally, aggressive discovery is an umbrella term imparted upon the exploitation of specific attributes of the member overlay protocols, such as leaf tables or peer lists, for discovery purposes. Hence, [187] offers no specific procedures for the protocol. Both message embedding and active notifications are opportunistic discovery methods and do not guarantee the timely discovery of nodes. Likewise, the peer exchange mechanism is similar to the one already in place in the modified Chimera protocol and would deliver no added benefit. Lastly, aggressive discovery, being a generic term, lacks any concise definition or specific procedure that would be useful for the implementation.

Unfortunately, other implementations from the field of P2P networks as cooperative systems either operate in a broadcast medium [177], use the mechanisms of the

Factor	Explanation
"Opportunistic Caching"	A single multicast response may update all nodes in a network reducing the volume of queries in a network.
Query Suppression	If several machines have the same query, only one device needs to transmit it, and yet, all nodes receive the response.
Passive Failure Detection	If a node observes an unanswered query, this information may be used to prune stale data from the cache.
"Passive Conflict Detection"	Multicast advertisements allow all nodes to promptly detect violations to required unique attributes i.e. peer keys.
Constrained Devices	Multicasting reduces the need for resources for response transmissions. These would otherwise be required to accommodate a list of destination nodes for each response.
Multiple Subnets	If a node receives an advertisement published in a subnet, multicasting guarantees that the advertised services exist on the local link, regardless of the source address.
Robustness	In the case where every node's address, default gateway, subnet mask, and DNS server addresses are incorrectly configured, the use of a multicast address ensures that all peers will still be able to receive advertisement on the local link.

Table 3.10: The advantages of mDNS [58].

aforementioned protocols [182, 183], do not offer mechanisms for discovery [153], or give incomplete solutions to the application's needs [185]. For these reasons, the next subsection will look investigate the application of flexible dedicated service discovery frameworks that may be tailored to the system. Specifically, due to the local nature of the infrastructure, a multicast-based framework may be used in favour of a unicast-based scheme to reduce traffic. However, out of the various multicast-based discovery frameworks available, mDNS and DNS-SD are considered as a possible solution because of the benefits listed in Table 3.10 and sourced from [58], which guarantee a system with low overhead.

3.4.2 The Management Service

As is typical of SOA-governed designs, the orchestration of the various services is a requirement. In this architecture, this feature is handled by the management service, which is both responsible for the transfer of information between services local to a device and those occupying the role of a node's decision engine. The former, communication between services, is done using UNIX domain sockets. The management service is a necessary intermediary between these services to allow it to decode received information and employ it in its decision making process. Currently, the algorithms that would be involved in such a process are outside of the scope of this work. Instead, the entire system operates in the manner shown in Fig. 3.21. After the management service is initialised, bi-directional sockets are created and the mDNS and P2P services are executed. After execution, the mDNS service initialises an mDNS discovery thread and collects all advertisements published by other nodes on its connected subnets, packages

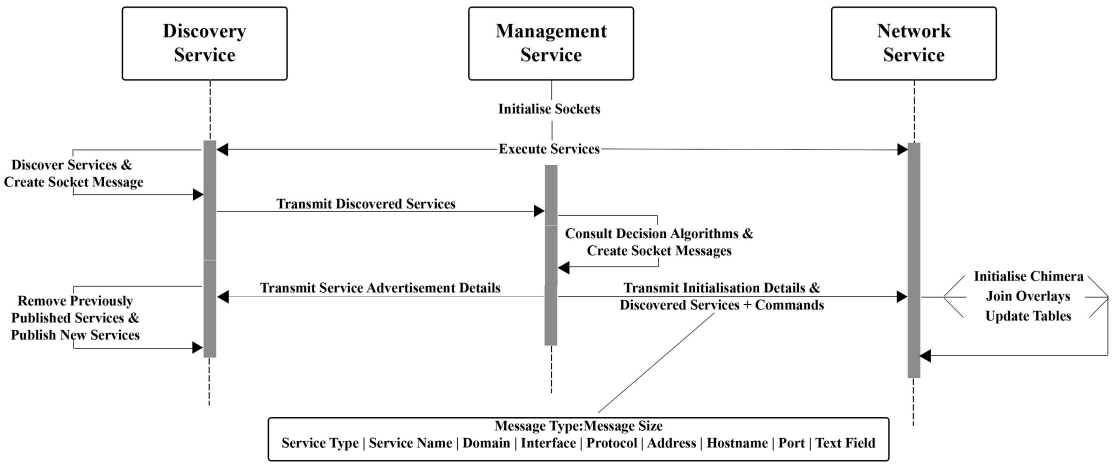

Figure 3.21: Interactions in the basic service set.

them in a string that is preceded by the message type and size, as shown in Fig. 3.21, and forwards the string to the management service via the socket. The management service decodes this information and decides which services the mDNS should advertise and what information to initialise the P2P service with. The service then creates messages identical in format to that shown in Fig. 3.21 for each service and forwards them accordingly. Once the information to be published is received by the mDNS service, the message is decoded and the services are published. If services are already being advertised by the mDNS service, these are removed and the new ones are published in their place. The P2P service also decodes the message to receive its key, its overlay key, the port it should run on, the overlays it should join, and the nodes the P2P service should add or remove from its routing tables. This process repeats indefinitely allowing published services and the status of the P2P service to be updated when required. The management service also executes an mDNS reflector during the initialisation process if it recognises that it is operating on a multi-homed device. With the entire framework detailed, the next section will proceed with its evaluation.

3.4.3 Experiment

The designed framework and the newly adopted discovery mechanism are tested using 11 Debian VMs hosted on a Xen Project (TM) server. These are distributed equally across two different network interfaces, with 5 nodes per interface and one multi-homed with access to both interfaces. The topology used is shown in Fig. 3.22. The experiment is executed on all 11 nodes sequentially using a BASH script running in Domain-0 of the server. After execution, the script pauses for 60 seconds during which the behaviour of the networking, routing, messaging, discovery, and other application layers and services are logged. Once the timer expires, the script resumes and terminates the framework

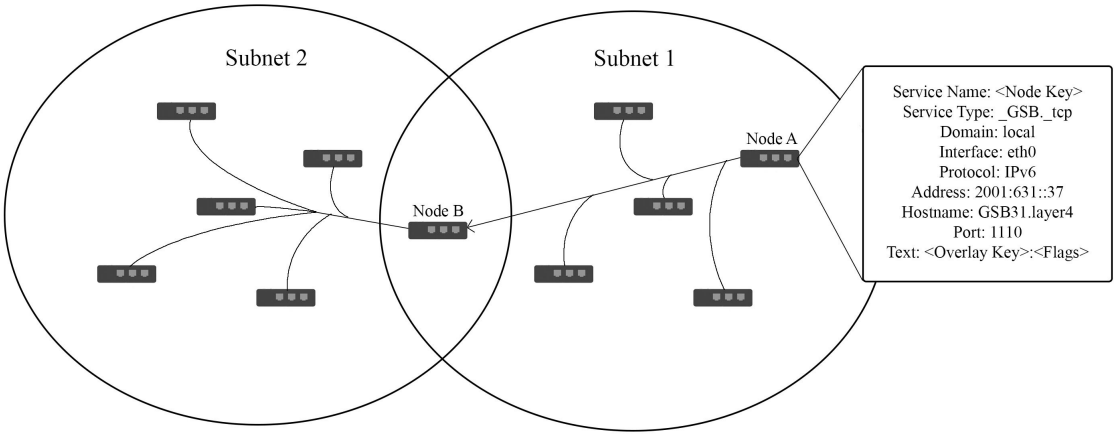

Figure 3.22: Transmission of mDNS advertisements in a distributed deployment.

on all nodes in sequential order before collecting all logs and restarting the experiment. The evaluation done here is to determine the speed of discovery within the context of the newly instantiated mechanism and architecture. The experiments were run a total of 33 times to ensure consistency of results.

For all 33 experiments, the discovery times for all nodes are summarised as a percentage distribution in Table 3.11 and the mean is plotted in Fig. 3.23. The time taken to discover a service is calculated from the point that the discovering node is initialised or the discovered node's services are published, depending on whichever occurs at a later time, until the discovering node receives a copy of the advertisement. As may be seen from Table 3.11, the discovery service allows all nodes to consistently discover any advertised services within 3 seconds from the time of publishing. Having therefore achieved its set goal of timely discovery, the next section will discuss opportunities for future work.

	% of services discovered by time T		
Experiment #	T=1s	T=2s	T=3s
1	49.59	100.00	100.00
2	59.50	99.17	100.00
3	52.89	100.00	100.00
4	42.15	93.39	100.00
5	64.46	100.00	100.00
6	53.72	91.74	100.00
7	38.02	97.52	100.00
8	37.19	92.56	100.00
9	53.72	97.52	100.00
10	53.72	100.00	100.00
11	30.58	92.56	100.00
12	33.06	95.04	100.00
13	36.36	95.04	100.00
14	36.36	92.56	100.00
15	43.80	98.35	100.00
16	34.71	90.08	100.00
17	42.98	98.35	100.00
18	61.16	99.17	100.00
19	38.02	97.52	100.00
20	53.72	94.21	100.00
21	54.55	99.17	100.00
22	56.20	100.00	100.00
23	57.02	99.17	100.00
24	55.37	100.00	100.00
25	43.80	98.35	100.00
26	55.37	99.17	100.00
27	53.72	98.35	100.00
28	39.67	96.69	100.00
29	48.76	96.69	100.00
30	47.93	95.87	100.00
31	41.32	96.69	100.00
32	38.02	95.87	100.00
33	57.85	99.17	100.00

Table 3.11: The percentage distribution of time to discovery in the experimental assessment.

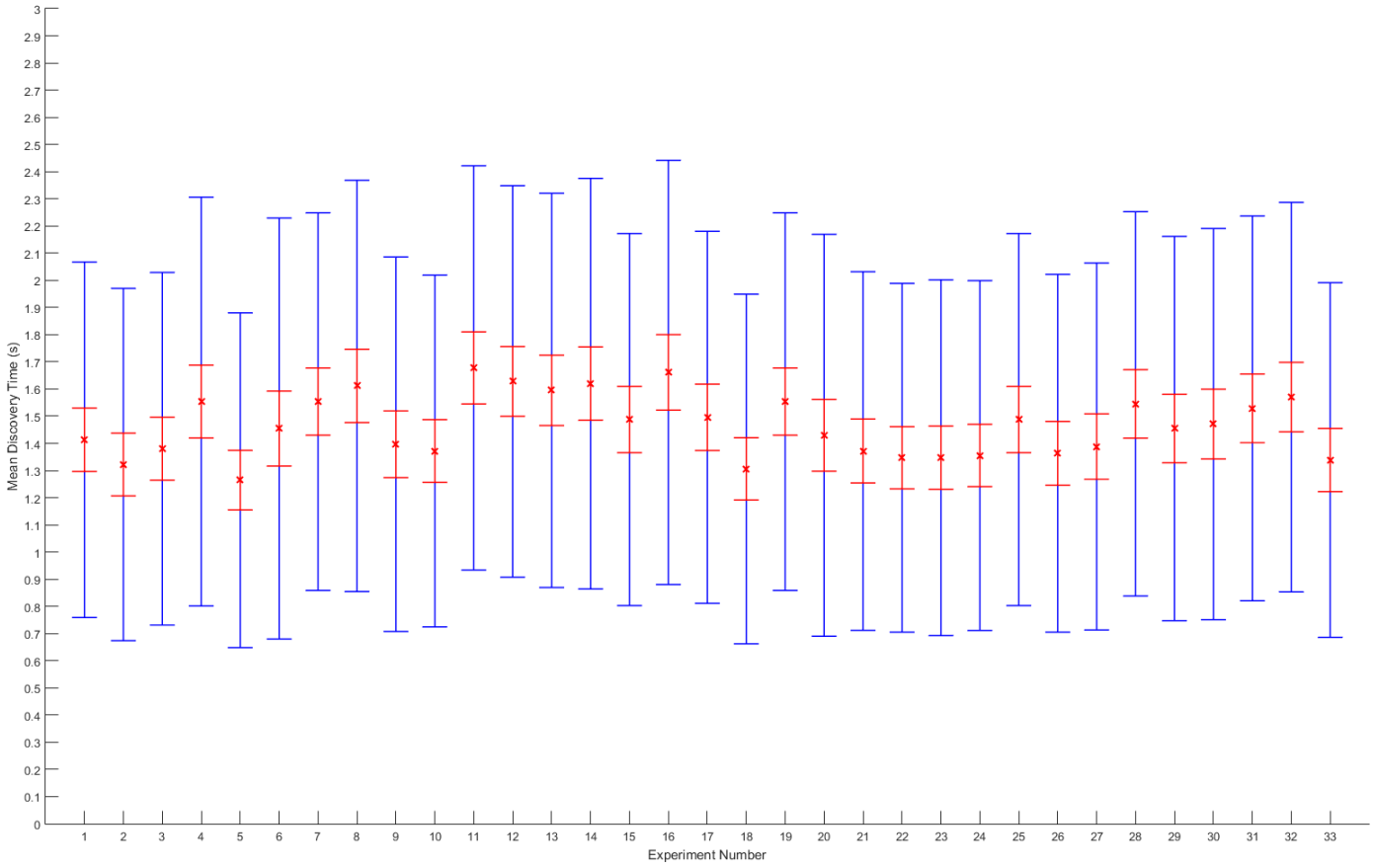

Figure 3.23: Mean discovery times with the standard deviation (blue) and standard error of the mean (red) for each experiment.

3.4.4 Discussion

This section is concerned with discussing possible enhancements for the discovery service as well as its relationship with previous work done in large industrial SOA research projects.

Potential Enhancements

Although the system achieves its goal of providing the underlying infrastructure required of a distributed system, improvements are, as always, possible. This section discusses two main concepts that may be applied to enhance the discovery service. Primarily, this involves modifications to the service advertisement change-tracking methods and the reflector implementation.

The detection of changes to advertised services, interestingly, includes the tracking of service advertisement removal. As was previously stated in Section 3.4.1, in the implementation, a node typically removes its published services before it advertises a new set received from its management service. This removal is done via a multicast message that may be integrated into the node as a method for the detection of failures in the network. Effectively, this mechanism may be used to replace the unicast heartbeat-based failure detection methods of Chimera with a multicast system that, again, would provide more timely detections of failure. Such a modification would allow for the integration of more enhanced fault tolerance mechanisms in the distributed system.

For example, in this implementation only one reflector may bridge the same set of networks. If more than one is active in the same subnets, a routing loop may occur that would allow for the amplification of multicast messages. This would effectively be a denial of service (DoS) attack on the bridged networks. Backups to the reflector node are necessary because the failure of a reflector node would be detrimental to the discovery process of the distributed nodes. Naturally, the heartbeat messages mechanism of Chimera may discover the failure of the reflector node. However, its opportunistic nature provides no guarantees in terms of timely detection. The implementation may be modified to incorporate a pinging mechanism exclusive to the reflector nodes, albeit at the cost of increased complexity. Instead, the service removal multicast messaging system that is already available may be used as an indicator for the detection of failed nodes and for the timely execution of appropriate measures by backup nodes in order to guarantee a responsive and survivable infrastructure.

A service removal mechanism in the manner described may be useful only in cases where failing nodes have ample time to publish service removal messages. This may not be the case, for example, if a node suffers immediate and total power loss. In this situation, techniques, like the use of non-maskable interrupts and large capacitors in a topology such as in Fig. 3.24 may be used to give the node enough time to notify the network of its impending failure. In the case of high load nodes, such as servers, UPS systems may be required instead of capacitors. To keep costs down, modules similar to server Intelligent Platform Management Interface (IPMI) subsystems, which would not have high power requirements, may be given the responsibility of monitoring for power

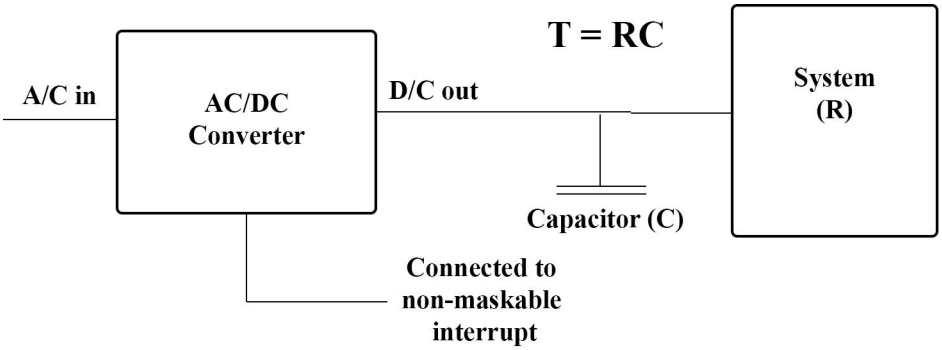

Figure 3.24: A mechanism for detecting and delaying the loss of input power.

outages and the publishing of service removal messages on behalf of high load nodes. To enact such a comprehensive system, however, requires a complete study of the methods available, costs involved, and mechanisms needed to guarantee timely transmissions of service removal messages in the various node and system failure scenarios possible.

A second possible method for improving the discovery service involves altering the reflector service to impart further reductions in network traffic. In this case, the reflector node, instead of forwarding all received mDNS messages to all other interfaces, may aggregate all of the resources in the received advertisements of one interface and publish them as resources local to the reflector node on all other interfaces. Mappings to the actual locations of these resources may then be kept by the reflector node. Thus, allowing it to route any subsequent attempts to access the advertised resources to their respective locations. Effectively, this would reduce the amount of messages being relayed between networks. However, in order to avoid a scenario where the reflector node, being the only reflector and hence aggregator, would suffer the fate of also being the only traffic relaying node between subnets, a method for the distribution of the services to be aggregated across multiple reflector nodes may be required. Alternatively, the reflector service may simply cache responses from one subnet and respond on behalf of those nodes to any polls it receives from its other interfaces, thereby reducing the impact of polling on large scale, multi-subnet deployments using a low-effort solution. A feasibility study is needed to determine the overhead cost of such approaches or modifications in comparison to that of the currently implemented technique of reflection.

On Contemporary Projects

Approaches implemented in large contemporary projects for SO industrial automation almost exclusively use WS-Discovery for zero-conf discovery. This include the IMC-AESOP, PLANTCockpit, and IoT@Work projects previously discussed in Chapter 2. WS-Discovery uses multicast SOAP-over-UDP messages for advertising, searching, and locating services on a local network. Similar to mDNS, the caching of multicast advertisements to reduce the number of subsequent in-network probes is encouraged. However,

a difference lies in the number of possible situations that support caching. For example, while mDNS stipulates that query responses be multicast, WS-Discovery requires unicast responses. This means that in mDNS, if several nodes have the same query, only one node needs to probe for the service and all nodes may benefit from the response. In contrast, with WS-Discovery's unicast responses, this is not possible and indicates that the two specifications may have different impacts on the network performance rates.

Another feature of WS-Discovery that may possibly impact performance lies in its message structuring technologies. The WS-Discovery specification requires the use of XML, while mDNS typically uses DNS-SD. In [68], the authors show that a 'Hello' multicast message predominantly containing addressing information is sized at 1088 bytes. Whereas packet captures from the Xen server showed mDNS query responses with frame sizes in the range of 300 bytes. Remedies are however possible, as [190] and [191] show that extensive savings in message sizes may be incurred if the EXI specification is applied to reduce the XML overhead of WS messages and possibly result in message sizes that are comparable with mDNS.

Without continuing with a granular comparison of the two specifications, it is already apparent that WS-Discovery, mDNS, and their companion technologies all target the same issues and, at times, use very similar techniques. However, differences do exist. Thus, there is a need for a detailed analysis and evaluation of the two systems to compare them in terms of their delivered benefits and expected performance. This is to determine the appropriateness of each as a solution for discovery in industrial environments.

3.5 Conclusion

An inter-system routing protocol is built based on the constraints of developing for industrial, SOA-governed, distributed systems. The system was subsequently deployed on 50 VMs on 64-bit and 32-bit platforms. Extensive testing was performed to determine the performance of the protocol and the optimal configurations for the organisation of Chimera peers.

The test results unearthed drawbacks that include high traffic loads and slow discovery times. Thus, the work was further extended by re-designing the networking layer as a basic service set for the non-RT coordination of industrial, SO, and survivable infrastructure. The resulting architecture applies network, discovery, and management services. It also uses sockets for inter-service communication, P2P technologies for inter-node communication, and mDNS and DNS-SD for discovery. The implemented framework is tested using 33 consecutive runs on 11 VMs across two subnets. Results showed that the presented services and their associated mechanisms consistently allowed for network-wide discovery of published services within a span of 3 seconds.

Further enhancements for the existing services are discussed. These include the use of service removal mechanisms, service aggregation and caching to reduce the impact of the proposed system on channel performance rates. However, based on the acquired results, the architecture is both stable and suitable for the failure-resistant relaying of messages in distributed manufacturing systems. The presented system may therefore progress

beyond the basic service set and incorporate advanced services for the acquisition, tunnelling, translation, storage, processing, and distribution of data. In the immediate sense this would involve the application of the services as a transport layer for middleware standards, e.g. OPC UA, as well as the extension of the SOA with capabilities to support hard RT tasks, as done in [192], to allow for the complete assimilation of field-level manufacturing components. This system would be a comprehensive solution capable of resolving age-old and complex challenges to vertical integration, system reconfiguration, and distributed execution in heterogeneous manufacturing environments.

CHAPTER 4

Enhancements to the Open-Platform Communications Unified Architecture

Several passages in this chapter are quoted verbatim from the following publications:

1. Ahmed Ismail and Wolfgang Kastner. Coordinating Redundant OPC UA Servers. In 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). September 2017, 1-8.

2. Ahmed Ismail and Wolfgang Kastner. Throttled Service Calls in OPC UA. In 2018 IEEE 19th International Conference on Industrial Technology (ICIT). February 2018, 1-6. In press.

The previous chapter developed alternate communication mechanisms for traffic engineering in networks of cooperative systems. This chapter continues to develop the envisioned system of flexible, agile, and resilient SO CPPS by designing and implementing enhancements for an OPC UA governed system of services. While OPC UA provides a complete technology stack, standardised interfaces and components, and guidelines, OPC UA also has gaps that are purposely left open for vendor-specific implementations.

For example, a SCADA system based on OPC UA is typically a large distributed system. It is "a system comprised of multiple software components running independently and concurrently across multiple physical machines" [30]. As noted in [193], large distributed systems need several forms of coordination. These include requirements for configuration, replication, synchronisation, group membership, discovery, leader election, and barrier synchronisation (resource fencing and locks) [193, 31]. For instance, OPC UA Servers running in warm redundancy failover mode need mechanisms for address space synchronisation and to ensure that only one server in the redundancy set connects to a downstream device at a time [133].

Since the need for coordination mechanisms is common across many distributed applications, Yahoo! created and open-sourced a generalised service for coordination,

named ZooKeeper. This service, with its uncomplicated filesystem-like API and strong guarantees for consistency, ordering, and durability has become the de facto solution for distributed coordination. The first goal in this chapter is to demonstrate the coordination of OPC UA server redundancy sets using Apache ZooKeeper [30].

The second part of this chapter addresses the client-server communication model in OPC UA. Communication between clients and servers in OPC UA is in the form of service calls. Effectively, these are Remote Procedure Calls (RPC). This implies that OPC UA predominantly operates using a client-side push-based communication model. The problem with this model is that it is possible for a single OPC UA Server to receive too many concurrent requests. In trying to process these requests, the server may exhaust or over-extend its available resources. Thus, the server may enter a failed or degraded state causing clients to experience high latencies, request time-outs, or service unavailability [194]. The degraded operation or unavailability of a component in an online manufacturing system is highly undesirable. The National Institute of Standards and Technology (NIST), for example, recorded an incident where a hanged control system in a wafer fabrication plant caused a financial loss worth US $ 50 000 [195].

It can be argued that the use of OPC UA Server *Redundancy Groups* and *ServiceLevel* indicators may alleviate the symptoms of server overload or prevent them from occurring. The former, involves sets of redundant servers with access to the same underlying resources and a synchronised information model. The latter, is a byte variable included in every OPC UA Server address space. It can have a value of 0, 1, 2-199, or 200-255 signalling that the Server is in a Maintenance, NoData (failed), Degraded, or Healthy state, respectively. According to the specifications, clients connected to a Degraded Server should not expect reliable services. Consequently, the client is permitted to switch to another healthy Server, if one is available. The client may also connect to multiple degraded servers to maximise its access to the underlying devices and their data. If connecting to a healthy server, a client is expected to select the one with the highest value. The sum of these tools therefore amounts to capacity planning and a simple load balancing strategy [133].

However, since capacity planning and load balancing do not alter the client-side push-based communication model of OPC UA, servers continue to be vulnerable to overload. In fact, as clients are permitted to switch from degraded to healthy servers, the situation may worsen as failures cascade across an impacted redundancy set and possibly cause the entire service to fail.

Other possible solutions limit the number of requests that are received and/or processed by an OPC UA Server. Thus implying rate throttling and load shedding, respectively. Both of these strategies involve the use of queues. Rate throttling would use a resourceful mediator to shield an OPC UA Server from traffic bursts. In contrast, load shedding operates on the premise that rejecting a service call uses significantly less resources than processing it. By limiting the number of requests being concurrently processed and rejecting the rest, servers are believed to reduce their chances of failing [196]. While both may be viable solutions to the problem, neither approach has been investigated in this context. Thus, this chapter also takes the first step by examining

the use of a rate throttling mediator to combat server overload in OPC UA. Similar to the coordination service, this is demonstrated using the Apache ZooKeeper service.

This chapter is therefore structured as follows. Initially, a primer on Apache ZooKeeper is provided. This is followed by an overview on OPC UA server redundancy. Next, the first deliverable, the coordination service, is detailed giving a description of the architecture, data model, and components used. The proposed system is evaluated through a prototypical implementation. The merits, limitations, and caveats involved in the system are discussed. Next, the queuing service developed for service calls rate throttling is similarly described and implemented. The code for both services is open sourced on Github[1, 2].

4.1 ZooKeeper

This section presents an overview of ZooKeeper by discussing its data structure, architecture, sessions, ZooKeeper Atomic Broadcast (Zab) protocol, local storage, failure resolution, and security [30].

4.1.1 Servers

The ZooKeeper service operates using a server-client architecture. Applications use a client library to interact with the ZooKeeper servers, as shown in Fig. 4.1. The servers can run in either standalone or quorum mode. The former is a single server and no replication of the ZooKeeper's state occurs. In quorum mode, a group of servers, termed the ensemble, work together to replicate the ZooKeeper state and serve client requests. A quorum is the smallest number of servers out of the ensemble that are needed to allow ZooKeeper to work.

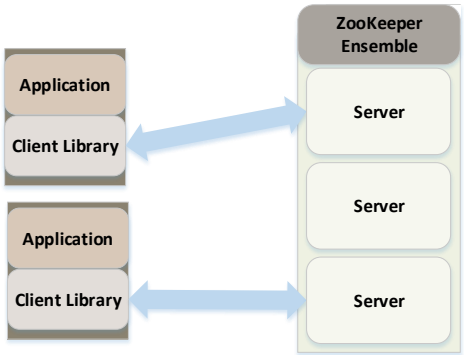

Figure 4.1: The ZooKeeper client-server architecture.

[1] https://www.github.com/AGIsmail/zkUACoordination
[2] https://www.github.com/AGIsmail/UaRateThrottling

Within an ensemble, a ZooKeeper server can be either a leader, a follower, or an observer. The leader is an elected position that manages all requests for state changes, including the ordering of changes. The leader transmits state changes as proposals that are voted on by the followers to ensure the replication of state changes across the quorum. Observers do not vote on state changes and only replicate committed updates. Observers are typically used to scale the system.

4.1.2 Sessions

Clients connect to any single server using a TCP session. Clients send heartbeat messages to keep sessions alive. Only a server can declare a session as expired. If, however, a client does not hear from its server for a certain amount of time, the session may move to another server. The server that it connects to is selected at random as a form of simple load balancing. Within a single session, requests are executed First-In-First-Out (FIFO). However, FIFO guarantees do not apply to concurrent and consecutive sessions.

4.1.3 Data Structure

ZooKeeper uses a hierarchical tree of data units, termed znodes, for its data structure. Znodes can be persistent or ephemeral. A persistent znode can only be removed through a call for deletion, whereas an ephemeral znode is also removed if the session of the client that created it expires. Because ephemeral znodes are session-dependent they are not permitted to have child znodes. Both persistent and ephemeral znodes can also be sequential. Sequential znodes are assigned unique sequentially incremented integers. An example data structure can be seen in Fig. 4.2.

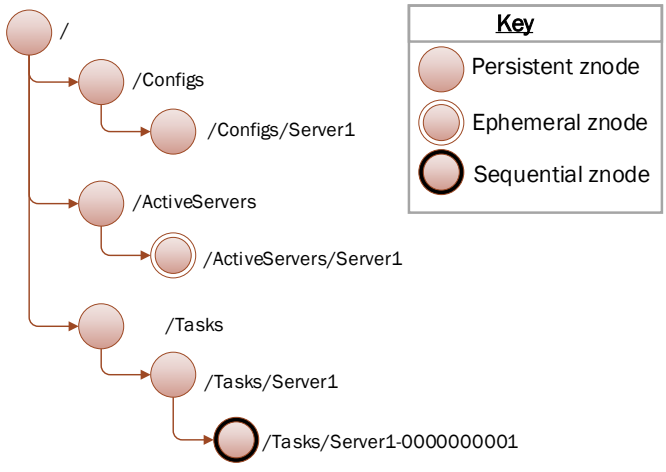

Figure 4.2: An example ZooKeeper data structure.

ZooKeeper supports the implementation of a quota on the number of znodes, called the count, and the size of data, called the bytes, that can be stored. Exceeding the set quotas only causes a warning to be logged and does not interrupt the operation of the system.

Every znode is given a version number that is incremented with every change to its data. This allows for the conditional execution of certain operations, such as deletion and data setting operations, as shown in Fig. 4.3. However, because a znode's version number is reset if it is deleted and re-created, conditional execution is not fool-proof. An example demonstrating the failure of conditional execution is shown in Fig. 4.4.

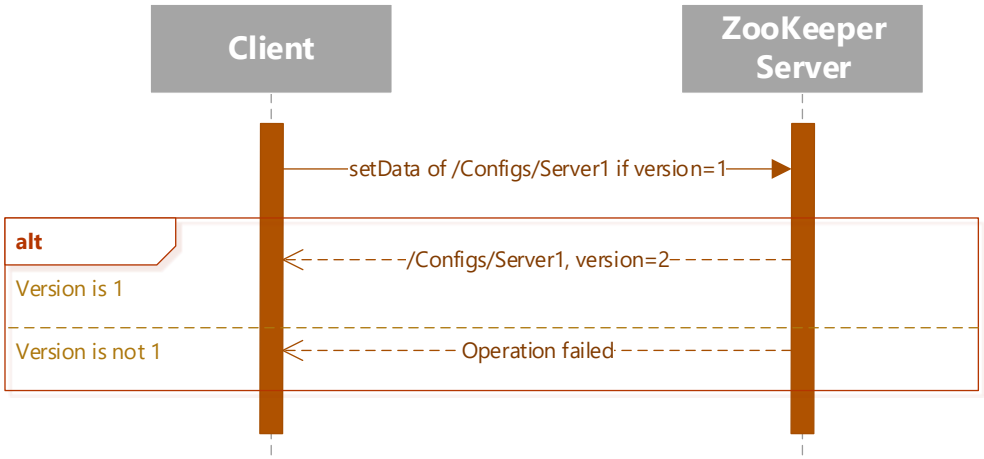

Figure 4.3: The coordination of concurrent updates using conditional execution. A data setting operation is dependent on znode */Configs/Server1* having a version number of 1. If the data set operation succeeds, the znode's version number is incremented to 2.

4.1.4 Watches and Notifications

Due to the performance penalty incurred by polling mechanisms, ZooKeeper favours a method based on notifications. Clients register for a notification by setting a watch on ZooKeeper. A watch is removed if the notification is triggered. To continue monitoring the znode, the client must therefore reset the watch. To avoid missing changes between receiving a notification and resetting the watch, watches are set using operations that read the znode's state. This mechanism is demonstrated in Fig. 4.5.

Watches can be set to monitor for changes to a znode's data, its children znodes, or its creation or deletion. They are persistent across servers and can only be removed by being triggered or if the creating client's session expires.

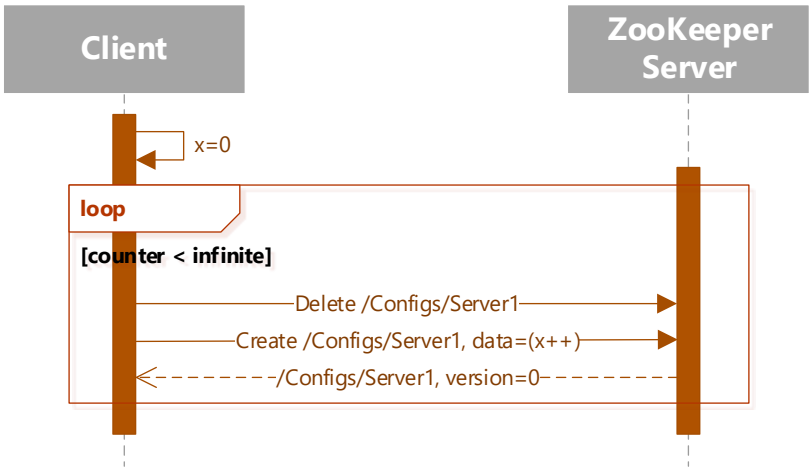

Figure 4.4: An example of a failure in conditional execution. The znode data is updated infinitely, but the version number remains the same.

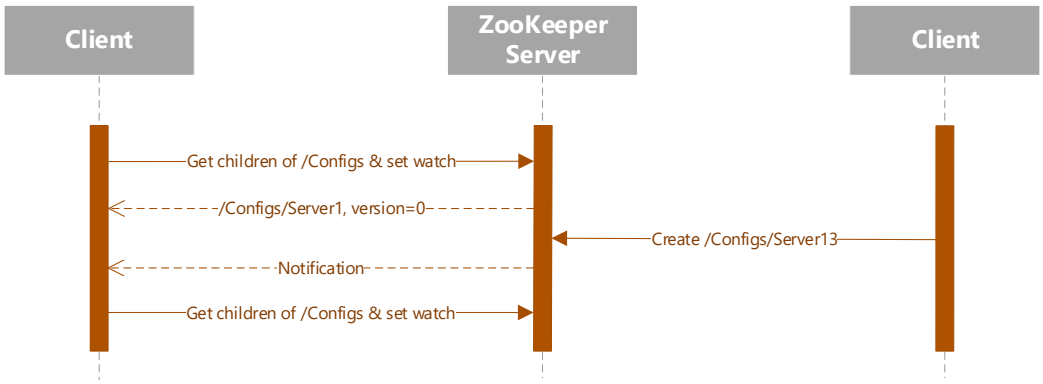

Figure 4.5: An example of a watch and notification on ZooKeeper. The watch set on the /Configs znode by the first client (left) is triggered by the creation of a new child znode under the same path by the second client (right).

Watches should be applied conservatively as they consume about 250-300 bytes of memory per watch, and the number of notifications sent for each watch is equal to the number of watches set for that znode. This rule of proportionality may result in undesirable traffic spikes on the network.

A final consideration related to watches relates to the 'Exists' watch, which is set to monitor for the creation of a znode. As a node's creation may be missed between the time that a client disconnects and reconnects, it should only be used for long-lasting znodes.

4.1.5 Requests and the Zab Protocol

Requests can be read requests or state changing requests. Both are atomic and either succeed or fail and no partial results are permitted. Reads are executed locally by ZooKeeper servers, while the writes are forwarded to the leader. The request is typically initiated by the client. The leader transforms the request into a transaction that describes the steps to be applied atomically and results in a state change. These transactions are committed using the Zab protocol.

The Zab protocol requires that the leader send the transaction to its followers as a proposal. The followers respond to the leader with an acknowledgement if they accept the proposal. Once the leader receives a majority of acknowledgements from the quorum it transmits a commit message to the followers and an inform message to the observers. This is shown graphically in Fig. 4.6. It is important to note that transactions are both idempotent and permanent. ZooKeeper does not support rollbacks.

Transactions generated by the leader are assigned a ZooKeeper Transaction ID (zxid).

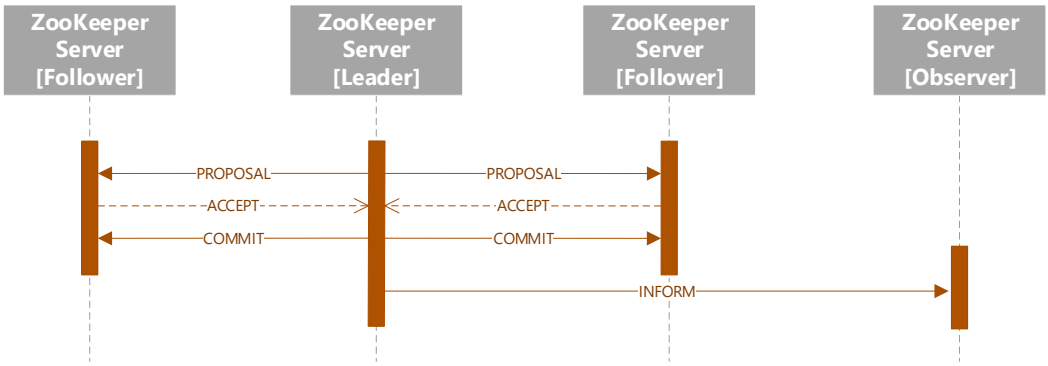

Figure 4.6: Simplified view of the message pattern between leader, follower and observer servers while committing a proposal.

Each zxid is a 64-bit integer used to ensure that transactions are applied in the order established by the leader, amongst other things. The zxid may be used for conditional execution.

4.1.6 Local Storage

ZooKeeper uses a pre-allocated transaction log file to persist ordered transactions on local storage. The transaction log is appended with proposals before they are accepted. ZooKeeper also offers snapshots, which are complete copies of the data tree serialised to file. Processes continue to execute during the snapshot taking process. This results in *fuzzy* snapshots that do not represent the true state of a tree at any specific point in time. However, this shortcoming may be remedied by replaying the transaction logs over the snapshot. Together, logfiles and snapshots may be used to recreate a server's state for later review or recovery.

4.1.7 Failures

Failures in ZooKeeper may occur in the service, network, or application and may be either recoverable or unrecoverable.

Recoverable failures, such as a network hiccup, are normal events and applications are written to continue running in spite of them. In the case of the leader, all actions should be suspended while in a disconnected state because no updates are received during this time. Clients, on the other hand, lose all of their submitted requests when they disconnect and need to resubmit them once they reconnect.

Unrecoverable failures are typically caused by expired sessions or an authenticated session no longer being able to authenticate itself. Unrecoverable failures, should be handled by exiting the application. Once the application starts again, it may resynchronise with ZooKeeper. This avoids the possibility of undesirable manipulations of data in multi-threaded applications that automate recovery.

4.1.8 Security

The security aspects of ZooKeeper include:

1. access control lists (ACL): Access rights are normally handled by the developer as they are set each time a znode is created. Access rights are not inherited by child znodes from their parent.

2. encrypted communication: Client-server communication may be encrypted if the server has Netty and Secure Sockets Layer (SSL) support. Quorum communication does not currently support SSL [197].

4.2 OPC UA Server Redundancy

This section provides an overview of redundancy features in the OPC UA standard. OPC UA supports the redundancy of both clients and servers to allow for availability, fault tolerance, and load balancing in different deployments. In OPC UA, this is achieved by allowing for duplicate instances of clients and servers. Special services, mechanisms, nodes, and client/server profiles are included in the specifications to support the various possible redundancy scenarios. These scenarios translate to a variety of failover modes that are available for OPC UA Server redundancy. A specific node, the *ServerRedundancy* node, is included in the address space to advertise the failover mode supported by an OPC UA Server. The different types of failover modes and their requirements are detailed in the rest of this section [70, 133, 24, 135, 198].

4.2.1 Transparent Redundancy

Redundant servers operating in transparent redundancy (TR) mode all run using an identical server URI and endpoint URL, as shown in Fig. 4.7. Therefore, the servers all appear as one server to connected clients. To allow a connected client to pinpoint the exact source of its data in a redundant server set, each server offers a unique ServerId. Information synchronisation between servers is the responsibility of the servers such that no actions are required by the client when a failover occurs.

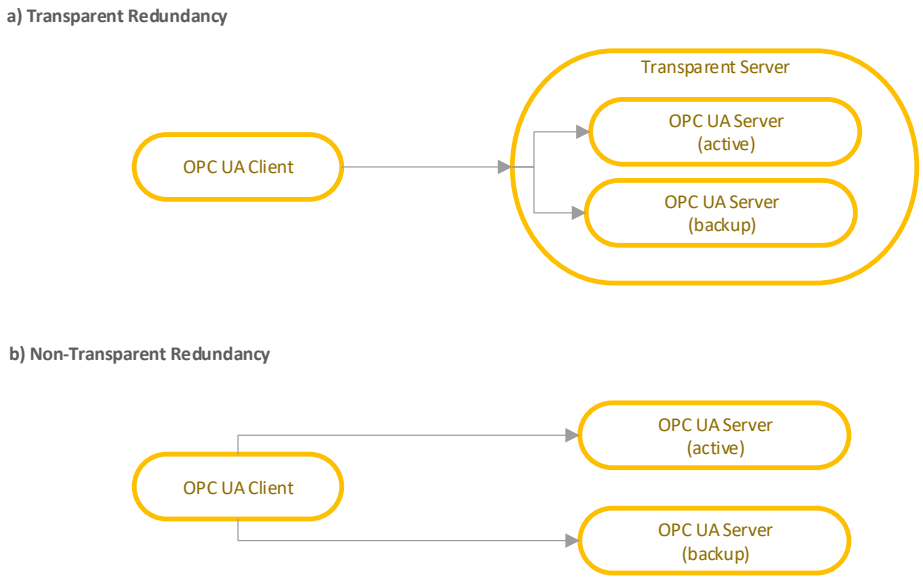

Figure 4.7: a) Transparent and b) non-transparent redundancy in OPC UA [133].

4.2.2 Non-Transparent Redundancy

In non-transparent redundancy (NTR) mode, clients are expected to participate in the failover. This typically involves selecting a server to failover to and moving session-relevant information to the new server. To do so, a server in NTR provides its clients with information on its failover mode and the other servers in its redundancy set. The different modes available in NTR are *Cold*, *Warm*, *Hot* and *HotPlusMirrored*.

Cold In Cold NTR, only one single server is active at a time.

Warm In Warm NTR, redundant servers may be active but only a single server can connect to the downstream device(s). This is useful in situations where the device(s) can only support a single connection at a time.

Hot In Hot NTR, all of the redundant servers are active and more than one server may be connected to the downstream device(s). The servers participating in a Hot NTR redundancy server set operate independently and are expected to have "minimal knowledge" of each other.

HotPlusMirrored In HotPlusMirrored, also referred to as *Hot+* and *hot and mirrored*, all of the servers in a group mirror their internal states across each other. More than one server may be active and connected to a downstream device. The mirroring must at least include "Sessions, Subscriptions, registered Nodes, ContinuationPoints, sequence numbers, and sent Notifications" [133].

4.3 Coordination in OPC UA Server Redundancy

Several aspects of server redundancy in OPC UA are in need of reliable coordination measures. This section presents these needs and details an integrated solution based on OPC UA and ZooKeeper.

4.3.1 Demands

The demands for coordination in OPC UA Server Redundancy include the following:

1. The first requirement is a synchronised address space such that an identical hierarchy of nodes is exposed to connected clients. This includes identical Nodes, NodeIds, browse paths, and address space structure. The only exempt nodes are the ones in the local Server namespace that, for example, expose server diagnostics information. This need is universal across all failover modes. Further requirements exist for specific failover modes, such as the replication of unique identifiers for events across servers in TR or Hot+ configurations [133].

2. The second requirement is a reliable mechanism for the detection of server failures. In the case of a TR server set, this would ensure the timely transfer of a session and its subscriptions to a functional server [133]. In NTR, this would allow for the automated initialisation of an application and/or connection to a downstream device. A service is therefore required to ensure failure detection and, if appropriate, contention resolution measures for the position of active server or to determine which server may connect to the downstream device.

The aforementioned requirements may be resolved by using ZooKeeper as a reliable configuration store for address space replication and as a central point for failure detection, leader election, and contention resolution. Using ZooKeeper for these services delivers several benefits to the system. The first is that any newly added OPC UA server only needs to be told how to connect to the ZooKeeper service and it may then download any other configuration information necessary, including its address space, and discern its role in the redundancy set [199]. The second benefit is derived from ZooKeeper's support for watches and notifications, which allow OPC UA servers to subscribe to changes and undergo run-time reconfiguration [199]. Third, ZooKeeper's snapshots and logs may be used to recreate the state of the information model at any point in time for diagnostic purposes. Last of all, ZooKeeper's other benefits include its guarantees for consistency, ordering, reliability, and its scalability born of its use of the Zab protocol and Observer servers.

To realise the above applications, certain requirements need to be imposed on the system:

- The entire address space of a redundancy server set must be stored on the distributed coordination service.
- Any modifications to the address space must be atomic and reflected on the coordination service.
- Any running server in a redundancy set must register its type and state on the distributed coordinator.
- For all of the above points, the distributed coordination service must be the only source of truth in the system.

To demonstrate this integration of OPC UA and ZooKeeper for distributed coordination, Subsections 4.3.2 to 4.3.4 present the overall architecture of the resulting system, the data model used for ZooKeeper, the different architectural components, and their implementation details.

4.3.2 The zkUA Architecture

The ZooKeeper-OPC UA (zkUA) system architecture is composed of four software components shown in Fig. 4.8:

1. ZooKeeper Ensemble: This is the distributed coordination service in the scenario. Out of an ensemble of n servers, the best practice is for n to be an odd number and that a majority be used to form the quorum. Doing so, would tolerate f servers crashing, where $f<n/2$, without it resulting in undesirable behaviour, e.g., split-brain scenarios [30].

2. zkUA Server: Every server participating in a redundancy set in the system must be integrated with ZooKeeper for several reasons. First, it ensures that any modifications to the address space locally or on ZooKeeper are reflected in the other. This allows redundant servers to provide connected OPC UA Clients with a homogeneous view of their address space. The integration is also needed for the correct operation of failure detection, leader election, and contention resolution for reasons that will be clear in Section 4.3.4.

3. zkUA Proxy: For the migration of existing OPC UA servers to the zkUA system, zkUA Proxies are required. This component is both an OPC UA and a ZooKeeper client. Its purpose is to replicate the address space present on an OPC UA Server to ZooKeeper. Replication of changes from an address space on ZooKeeper to legacy OPC UA Servers should not be supported as the legacy servers may include functions that would disrupt the overall behaviour of the zkUA system.

4. zkUA Failover Controller: This component is the main management component in the architecture. It is responsible for detecting and reporting zkUA Server failures, initiating a failover, and participating in contention resolution and resource fencing. The details of this component are discussed in Subsection 4.3.4.

4.3.3 The ZooKeeper Data Model

A fifth component essential to the zkUA system is the data model deployed on ZooKeeper that is also shown in Table 4.1 and Fig. 4.9. The first znode under the root node is the /Servers znode. This denotes the parent znode under which all OPC UA Server redundancy sets operate. Each redundancy set requires a neutral identifier under the /Servers znode where the set's address space may be stored and its activities organised. A Globally Unique Identifier (GUID) is currently used to represent each redundancy set. To advertise the GUID via the OPC UA address space, a new Variable node, the *GroupGUID*, is created under the OPC UA-specified ServerRedundancy object. The GroupGUID's value is set to the GUID of the redundancy set. Both the path and the value are shown in Fig. 4.10.

Every Node is uniquely identified on ZooKeeper under the /AddressSpace path using its NodeId. The NodeId is converted into a string that holds the Node's namespace and

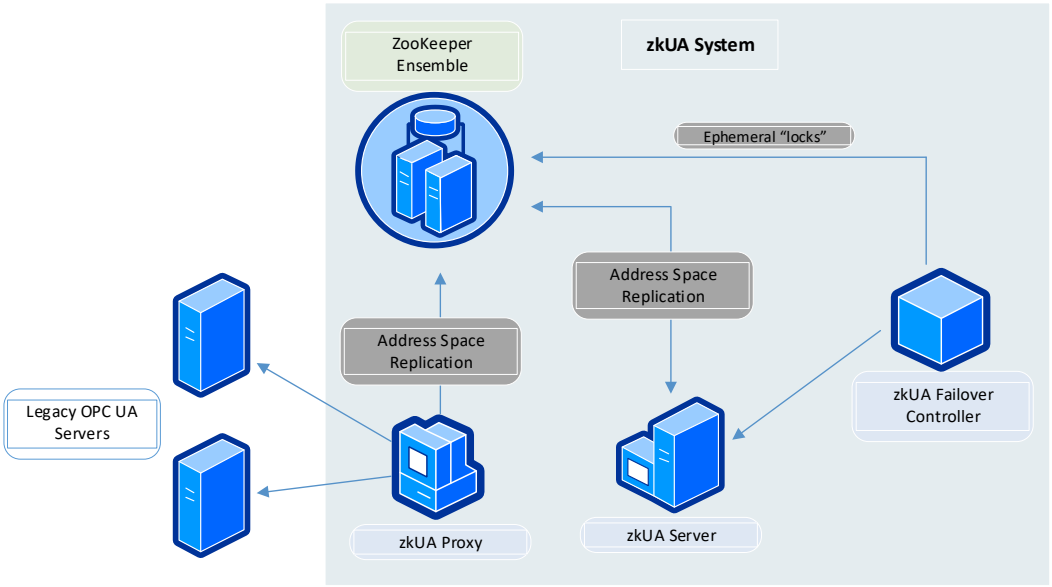

Figure 4.8: The zkUA system architecture.

ZooKeeper Path	Type of znode	Explanation
/Servers	Persistent	The root path for zkUA Server redundancy sets.
/Servers/{GroupGUID}	Persistent	Every redundancy set is assigned a unique GUID to differentiate it from the others.
/Servers/{GroupGUID}/AddressSpace	Persistent	This path stores all of the OPC UA Nodes with their respective attributes and references for storage, synchronisation, and replication.
/Servers/{GroupGUID}/Active	Persistent	If a zkUA Server is in a functional state, connected to a downstream device, and ready to serve clients, then it is in an active state and is represented by an ephemeral znode under this path.
/Servers/{GroupGUID}/{Failover Mode}	Persistent	All zkUA Servers that support a specific failover mode and are part of the same redundancy server set are each represented by an ephemeral znode under the correct Failover Mode path. Paths are available for Transparent, Cold, Warm, Hot, and Hot+ redundancy.

Table 4.1: The ZooKeeper data model for OPC UA servers redundancy.

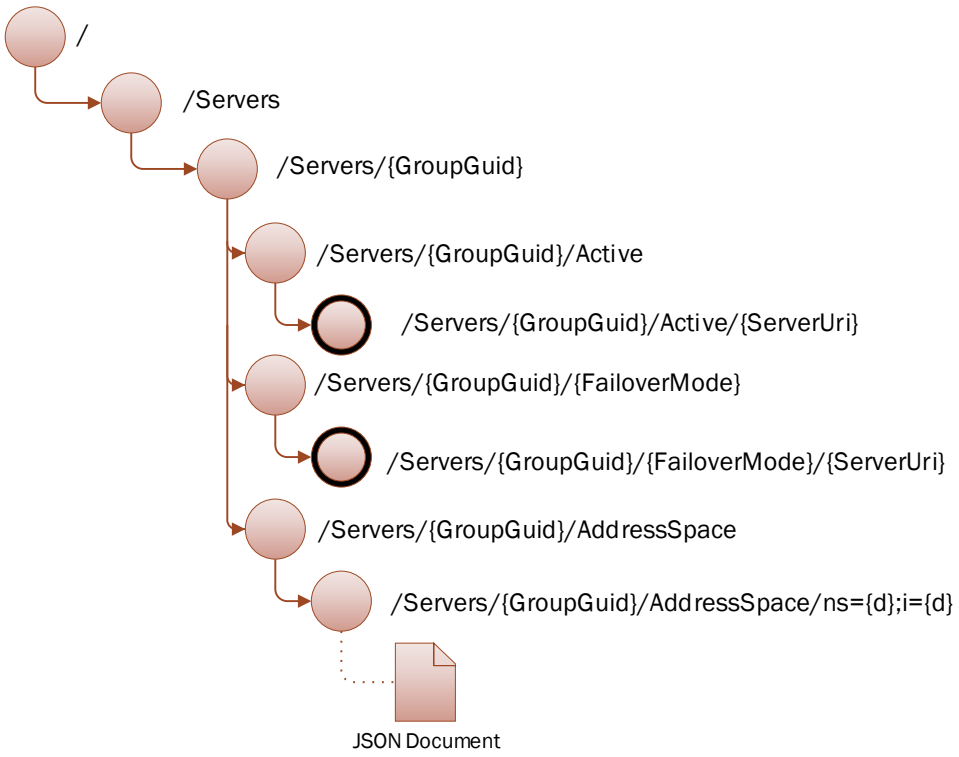

Figure 4.9: The hierarchical ZooKeeper data model for OPC UA servers redundancy.

identifier. For example, a node belonging to namespace index 1 NodeId 3000 would be represented as "ns=1;i=3000" under the AddressSpace path. The znode's data is set with the encoded attributes of the Node it represents.

Naturally, this data model may be extended based on the failover mode to accommodate any additional information requiring synchronisation across the redundancy set. While in its current state, however, it is apparent that an indirect result of having all zkUA Servers register on ZooKeeper is that ZooKeeper unwittingly doubles as a discovery service. The remainder of the data model will be clarified in the next subsection as the operations of the zkUA components are discussed.

4.3.4 The zkUA Components

zkUA Proxy The zkUA Proxy is developed as a migration path for non-ZooKeeper integrated OPC UA Servers. It is composed of a ZooKeeper client and an OPC UA Client. The proxy is initialised with a configuration file that specifies several parameters

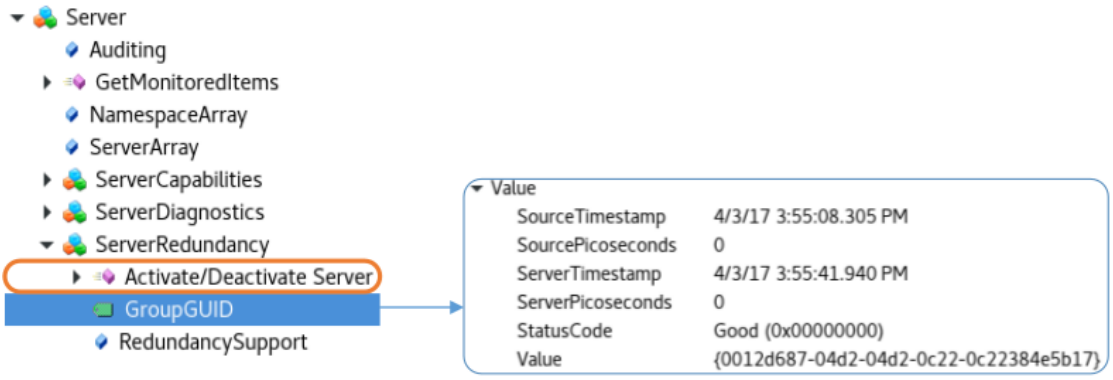

Figure 4.10: The "GroupGUID" variable node and "Activate/Deactivate Server" method node in the OPC UA address space.

Parameter	Component	Explanation
Hostname	Proxy/Server	The hostname belonging to the OPC UA Server.
PortNumber	Proxy/Server	The port number the OPC UA Server is bound to.
GroupGUID	Proxy/Server	The GroupGUID representing the UA redundancy group on ZooKeeper.
Username	Proxy/Server	The Username used to login to the UA Server.
Password	Proxy/Server	The Password used to login to the UA Server.
ZkServer	Proxy/Server	The hostnames and port numbers of ZooKeeper servers participating in the ensemble.
RedundancyType	Server	The zkUA Server needs to be informed of its redundancyType. Options are: standalone, transparent, cold, warm, hot, and hot+.
State	Server	Instructs the zkUA Server if it is to start in an active or inactive state.
ServerId	Server	Required if the server is running in transparent redundancy mode, otherwise it is not.
AvailabilityPriority	Server	If set to true, the locally cached address space is used as a fallback for reads in case the server's connection to ZooKeeper is interrupted.

Table 4.2: The zkUA start up configuration file parameters.

listed in Table 4.2. The proxy must be told which OPC UA Server's address space to replicate to ZooKeeper. It must also be told how to connect to it and the redundancy server set's GUID so that it may push the encoded address space to the correct path on ZooKeeper. A UML sequence diagram demonstrating the actions of a zkUA Proxy is shown in Fig. 4.11.

zkUA Server The zkUA Server, similar to the zkUA Proxy, is initialised using a configuration file providing it with start up parameters such as its URI, port, redundancy

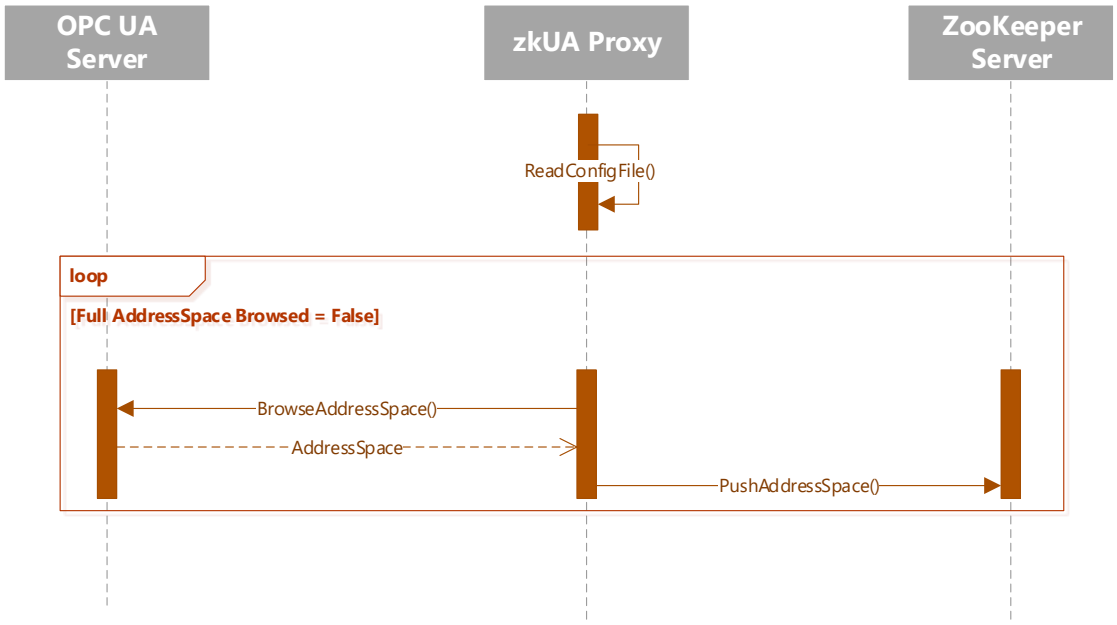

Figure 4.11: A UML sequence diagram showing the actions of a zkUA Proxy component.

group GUID, failover mode and role, as shown in Table 4.2.

An additional parameter, *AvailabilityPriority*, is included in the configuration file to allow reads to continue from the zkUA server's local cache even if the communication between it and the ZooKeeper ensemble fails. Although this stops ZooKeeper from being the only source of truth, it may be required in cases where the continued uninterrupted availability of the zkUA Server is necessary. In such cases, it is advisable that modifications to the local cache be kept at a minimum, e.g., only permitting the continued polling and storage of values from downstream devices in the address space while disallowing any add/delete operations or the modification of Node types until the connection to ZooKeeper is restored.

A new Method Node, the "Activate/Deactivate Server", is also needed by every zkUA Server. The Method Node and its associated internal function are used to modify the state of a zkUA Server when a failover is initiated.

The interactions that take place between a zkUA Server and ZooKeeper are shown in Fig. 4.12.

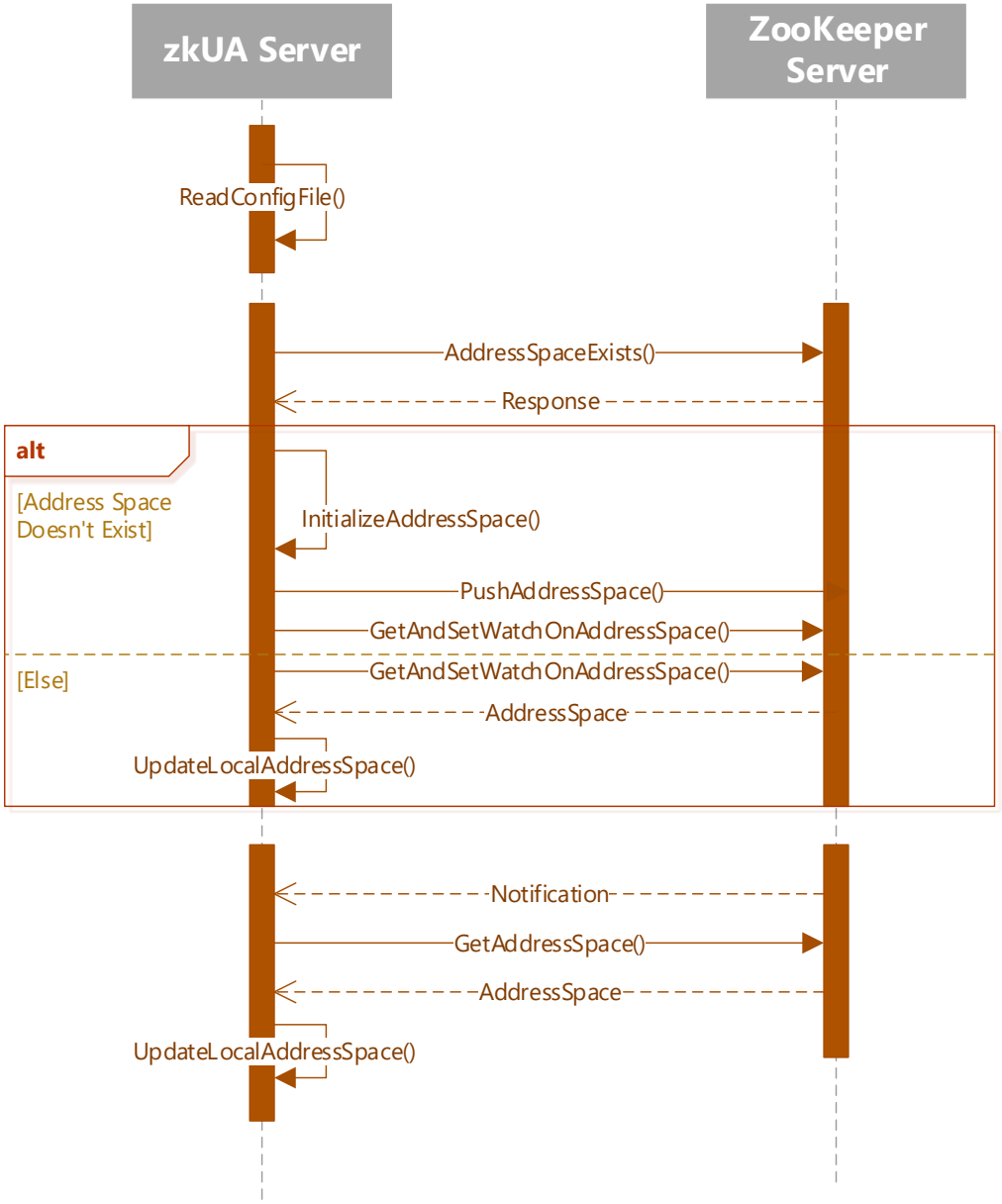

Figure 4.12: A UML sequence diagram showing the interactions taking place between a zkUA Server and ZooKeeper. The triggered notification implies that the address space stored on ZooKeeper was manipulated by a zkUA Server.

zkUA Failover Controller Finally, the zkUA FC is one of the most critical components in the system because it manages the behaviour and role of the zkUA Servers. The zkUA FC, similar to the proxy, is both an OPC UA Client and a ZooKeeper client. The zkUA FC is initialised with all of the parameters found in a zkUA Server configuration file except for the "AvailabilityPriority" parameter. These configurations are used by the zkUA FC to know which zkUA Server to monitor, how to connect to it, and what the correct failover mode and behaviour should be.

The zkUA FC is modelled on Hadoop's HDFS ZKFailoverController [200] and, therefore, performs the following tasks:

- zkUA Server registration: Once initialised with a specific failover mode, the zkUA FC opens a session with ZooKeeper and creates an ephemeral znode under the redundancy group's path for that mode.

- zkUA Server status monitoring: the zkUA FC of a specific zkUA Server periodically polls the server's state to determine its health. If the server responds in a timely manner with an acceptable state then it is considered to be healthy. Otherwise, it is not. The acceptable responses for the server's state differ with the failover mode.

- zkUA contention resolution: If a zkUA Server is initialised in an active state and the failover mode supports more than one active server at a time, the zkUA FC creates an ephemeral znode under the redundancy set's Active path. If the failover mode or scenario only supports one active server at a time then only one zkUA FC and server combination is capable of creating an ephemeral znode under the Active path at a time, effectively acquiring a lock for the downstream device. The zkUA FC/server combo that is first to create the znode acquires the lock. All other controllers then monitor the lock for deletion. This typically occurs if the zkUA FC with the lock determines that its zkUA Server is in an unhealthy state. In such a case, the zkUA FC terminates its session with ZooKeeper. As the session expires, the ephemeral znodes are deleted, and all other controllers monitoring the znode are notified of the deletion event. The respective active controllers then try again to be the first to create a lock.

The above actions are demonstrated using a UML sequence diagram in Fig. 4.13.

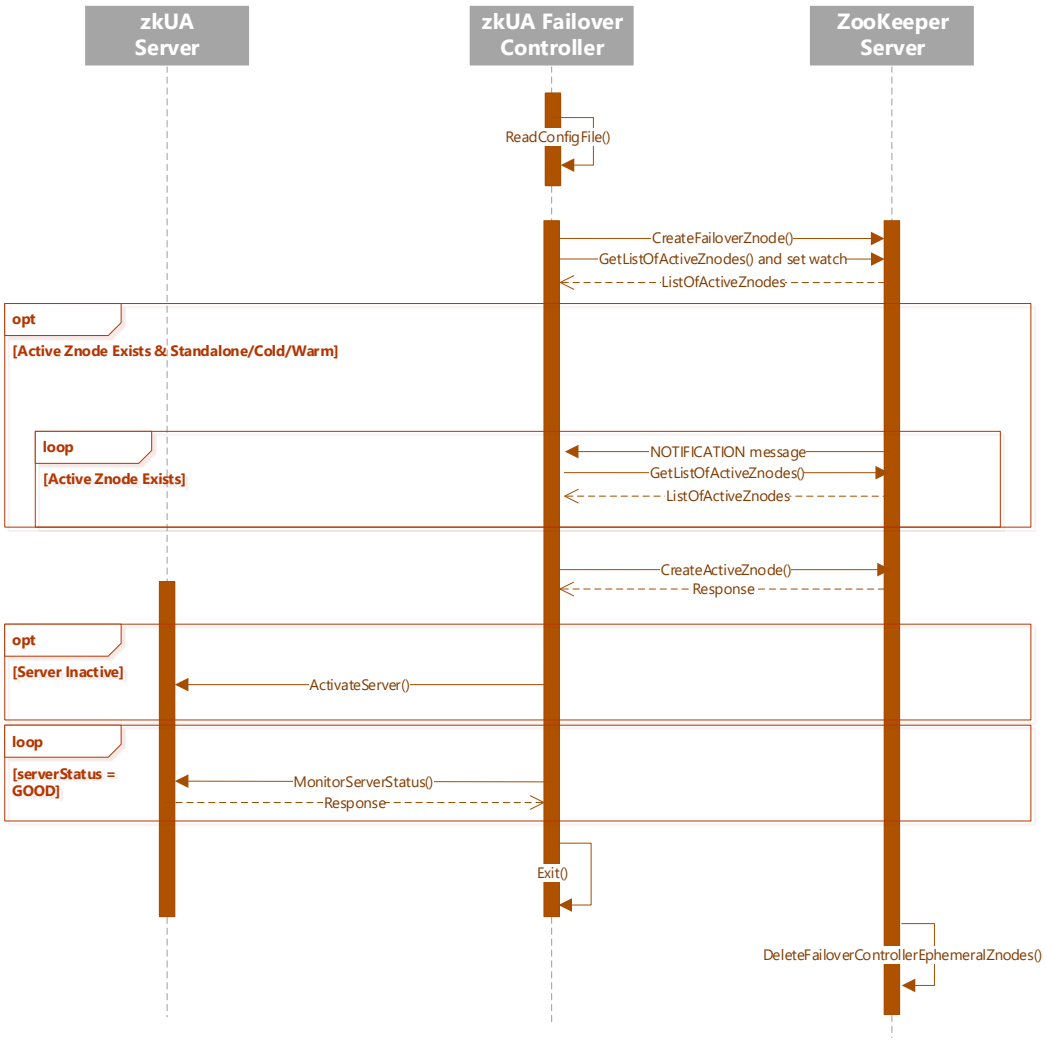

Figure 4.13: A UML sequence diagram showing the interactions taking place between a zkUA Server, a zkUA FC, and ZooKeeper.

4.3.5 Implementation

A prototypical implementation[3] is made using the open source ZooKeeper[4] and open62541 libraries [201]. The resulting code achieves the requirements in Section 4.3 by intercepting calls in the open62541 library to OPC UA Node addition, deletion, and modification (attribute writing) functions. Specifically, the functions are re-defined in the amalgamated open62541 library header file such that calls to the functions listed below are redirected to zkUA interception functions.

- Service_Write: This function is called once when an OPC UA Client modifies the attribute of a Node on an OPC UA Server over the network.
- UA_Server_Write: This function is called once when an OPC UA Server edits a Node's attribute.
- Service_AddNodes_single: This function is called once when an OPC UA Client or Server adds a new Node to the address space of the OPC UA Server.
- Service_DeleteNodes_single: This function is called once when an OPC UA Client or Server deletes a Node from the OPC UA Server's address space.

For attribute changing operations, the intercepting function should save a copy of the current state of the Node to be modified before calling the original open62541 function. Once the original function finishes updating the local cache, the intercepting function then encodes the Node's NodeId, attributes, and its parent's NodeId and reference and pushes the encoded information to ZooKeeper. The parent's NodeId and reference are required to preserve the structure of the address space when replicating to another OPC UA Server. If the push fails, then the entire process should be reversed. The same process applies for the addition of a new Node, except for saving a copy of the original state of the Node because it should not yet exist in the local cache.

When a Node is modified and pushed to ZooKeeper, the remaining servers in the redundancy set receive a notification for the change in the znode's data. Since the Node may already exist on the zkUA Server, the node must be deleted and re-added. This is because the Node's type may have been modified in the process thereby invalidating the option of using attribute writing functions during replication. For these cases, Node deletion must not be replicated back to ZooKeeper and must only be enacted upon the local cache to prevent unwanted behaviour in the system.

Every Node that is added or modified in a zkUA Server is accompanied by its transaction zxid. The Node's znode path and zxid form key-value pairs that are stored in a hashtable local to each zkUA Server. The hashtable is then used to guarantee that the local zkUA Server is in sync with ZooKeeper and to prevent unnecessary operations to the address spaces stored on the zkUA Servers.

[3] https://github.com/AGIsmail/zkUACoordination.git
[4] https://zookeeper.apache.org/

Read operations do not need to be redirected to ZooKeeper. By setting watches on the entire address space on ZooKeeper, the zkUA Server ensures that the local cache is always up to date. The only case in which this does not apply is when the zkUA Server's cache is being read while it suffers from an interrupted session. In such situations, reads from the local cache are forbidden unless the "AvailabilityPriority" parameter is set to true in the server configuration file for the special case where the availability of the zkUA Server is more important than the reliability of the served data.

As may be expected, further details related to the failover modes and other implementation specifics exist. However, the above suffices for the purposes of the discussion in the next section. The codebase contains further information[5].

4.3.6 Discussion

The presented architecture and accompanying implementation ensure that the entire address space of a redundancy server set is stored on ZooKeeper. Any modifications to the address space are atomic in nature. All servers are registered on ZooKeeper and participate in failure detection, leader election, and contention resolution. The system is designed to treat ZooKeeper as the only source of truth, thereby avoiding split-brain scenarios.

However, there are caveats involved in using such a system. First, while the scalability of reads are possible using ZooKeeper Observers, due to the per-watch-set memory penalty, the ZooKeeper service may require memory-abundant systems. Yet, since watches are stored locally on the ZooKeeper server that the zkUA Server connects to, this should not be a problem if the overall system is designed with enough ZooKeeper servers and hardware resources. The design should, therefore, reflect the expected overall number of server redundancy sets and their respective address space sizes.

A second point to address is security. As mentioned in Subsection 4.1.8, ZooKeeper provides support for ACLs. In principle, this should prevent any unwanted manipulation of zkUA redundancy set behaviour. The system may be hardened further if communication between the ZooKeeper servers and the zkUA Servers, Proxies, and Failover Controllers are encrypted with SSL. Since communication between the quorum is not encrypted, tunnelling may also be necessary.

Another point to address in this section is related to an unintended, yet positive, consequence of the presented architecture. Since the ZooKeeper service is effectively storing the address spaces of all of the participating zkUA servers, ZooKeeper may be considered an active OPC UA Chaining Server. However, to exploit this situation a specially designed zkUA Client capable of reading directly from ZooKeeper would be required. Otherwise, a proxy that is both a ZooKeeper Client and a shell OPC UA Server may act as an interface to ZooKeeper for other OPC UA Clients.

One other detail to discuss is Method Node replication. While not present in the prototypical implementation, possible solutions to achieve this may require the following.

[5] https://github.com/AGIsmail/zkUACoordination.git

1. The first solution would have the functions associated with the Method Nodes be available on all of the zkUA Servers in the entire system. This allows for a single generic zkUA Server to be used throughout the system. This may, however, result in a larger sized implementation which has a number of costs associated with it, e.g., a larger attack surface. Alternatively, each redundancy set may have its own flavour of zkUA Servers with the appropriate functions built. While potentially resulting in smaller implementations, this would increase the complexity involved in managing and developing for the system.

2. The second possible solution would use function stubs. An implementation in this case may follow a service-oriented approach whereby a sub-system of services is created to represent the different methods to be called.

As may already be apparent from the architecture of Subsection 4.3.2, the zkUA FC, while critical to the system, is a single point of failure. A zkUA FC may crash while its zkUA Server remains functional. This vulnerability is mirrored in Hadoop's HDFS [200]. For mitigation, the zkUA FC should be monitored for unexpected failures and restarted appropriately [200]. To ensure the orderly operation of zkUA Servers, a possible solution may include having active zkUA Servers watch for the disappearance of their ephemeral znodes on ZooKeeper and forfeiting their active roles when appropriate.

While several other papers have addressed implementations for OPC UA Aggregation Servers [202, 203] or Historical Servers [204], there appears to be only one other publication addressing OPC UA Redundancy [198]. In [198], a different goal is pursued as the paper carries out a performance evaluation for OPC UA redundancy using the Java Client-Server SDK by Prosys. While this makes the presented system unique, it also deprives it of a basis for comparison.

The coordination requirements of OPC UA Server redundancy have been quantified and shown to be realisable using the ZooKeeper service. The next sections will address the queuing service for rate throttling service calls in OPC UA client-server interactions.

4.4 Throttled Service Calls in OPC UA

This section examines the use of a rate throttling mediator to combat server overload in OPC UA. Subsection 4.4.1 defines the requirements for the queuing service and demonstrates how features of Apache ZooKeeper can be used to meet these requirements. Subsection 4.4.3 proceeds to describe an open-source prototype developed to evaluate a suitable architecture, data model, and communication flow for rate throttling based on ZooKeeper. Finally, Subsection 4.4.4 provides a discussion on the presented system.

4.4.1 Requirements

This section defines the requirements necessary of a mediator for the queuing of OPC UA service calls. These requirements are summarised in Table 4.3.

Parameter	Requirement
Scalability	The system should be able to scale to support 1000's to 100 000's of connected OPC UA Clients and Servers.
Consistency Guarantees	The service must reflect a consistent state.
Recoverability	The service should be able to recover from system and network problems.
Security	The system must, at least, provide the same security features as OPC UA.
Client-Push/Server-Pull Communication	OPC UA Clients must push to the queueing service, while OPC UA Servers must pull tasks off their respective queues when they wish to receive the data.
At Least Once Semantics	Every service call should be delivered to a server at least once.
Supportability	The service must be well-documented, actively developed, and have a healthy support community.

Table 4.3: The criteria for a task queuing service.

First, the queueing service is expected to be highly scalable. It should be able to support thousands to hundreds of thousands of concurrent connections from both OPC UA servers and clients. Scalability should, in this case, be horizontal due to the associated benefits [205]. This means that the tool should be able to operate as a distributed system.

For service calls, submissions should be committed in a transactional and strongly consistent manner. According to [206], this implies four properties listed below:

- Atomicity: An operation either succeeds or fails. Inconsistent states are not permitted.
- Consistency: "Committed transactions are visible to all future transactions" [206]. This means that all redundant participants in the queueing service apply transactions in the same order, thereby preserving the uniformity of the service state [207].
- Isolation: Uncommitted transactions are not visible to future transactions.
- Durability: Transaction commits are permanent.

These are important properties for task queuing in safety-critical environments. Partial or non-permanent commits, fuzzy reads, and dirty reads and/or writes imply that an online system may operate out of specification. Given the tight and complex coupling between software and physical processes in CPPS, the outcome may manifest as undesirable physical events.

A third point to address is availability and fault-tolerance. In an ideal system, every submitted service call should be successfully queued and subsequently made accessible to their respective OPC UA Servers. However, system and network problems are to be expected. Measures should therefore be included to tolerate crashed node faults. These would allow connected OPC UA Clients and Servers to continue using the queueing service in cases of partial system failures. The service should also have protections in place

against network partitioning. If the system does not protect against network partitions, split-brain behaviour may manifest. This situation may, again, have an undesirable effect on the manufacturing environment.

Naturally, clients must be able to submit tasks for execution with associated metadata. Likewise, servers must be able to retrieve them. At-least-once message delivery semantics are necessary to guarantee that each service call is consumed by a server. Messages should also be delivered to servers using pull-based mechanisms to mitigate the previously discussed server overload scenarios.

Moreover, the system should match, if not improve upon, the security features inherent to OPC UA. The system should therefore incorporate measures to ensure that the submission, retrieval, and modification of queued tasks are only done by authorised clients. This amounts to the inclusion of authentication and encrypted communication features.

The last point to discuss is supportability. The tool should have strong community and developer support available. It should be well-documented and actively developed to ensure a long and stable lifetime for the service.

The next subsection will demonstrate how the Apache ZooKeeper service meets these requirements, thus making it a suitable platform for the development of the queueing service.

4.4.2 Relevant ZooKeeper Features

This subsection highlights the relevant features and strengths of ZooKeeper that demonstrate its ability to serve as a queuing service for OPC UA. This discussion is based on the ZooKeeper overview previously given in Section 4.1. A summary of the required parameters and respective ZooKeeper properties is given in Table 4.4. Thus, ZooKeeper meets the demands outlined in the prior subsection as follows.

- **supportability** is considered a met requirement due to ZooKeeper's wide adoption, detailed documentation[6], and frequent updates[7].
- For **consistency guarantees**, the leader of a ZooKeeper quorum is an elected server that is responsible for executing and ordering requested state changes using the Zab protocol. This is a two-phase commit protocol that operates as follows (c.f. Fig. 4.6).
    1. A requested state change is transformed by the leader into a transaction that includes the steps needed to atomically apply the state change.
    2. The transaction is transmitted by the leader to its followers as a proposal.
    3. Each follower checks that the proposal is from its current active leader and that it conforms with the current order of acknowledged and committed transactions.

[6] https://cwiki.apache.org/confluence/display/ZOOKEEPER/Index
[7] https://zookeeper.apache.org/releases.html

Parameter	Features
Scalability	Quorum mode, observers, dynamic scaling.
Consistency Guarantees	Zab protocol, majority quorum in an odd numbered ensemble.
Recoverability	Zab protocol, transaction logs, snapshots.
Security	Client-server and server-server mutual authentication, client-server SSL, znode-level access control.
Client-Push/Server-Pull Communication	Write/read operations and watches.
At Least Once Semantics	Watches, persistent znodes, and read operations.
Supportability	Mature, widely adopted product [31].

Table 4.4: The features of ZooKeeper that meet the criteria for a task queueing service.

4. If the proposal is found to be compliant, then the followers accept the proposal and respond to the leader with an acknowledgement.
5. The minimum number of servers that need to respond to a proposal with an acknowledgement is referred to as the quorum. Once the leader receives enough acknowledgements, it sends a commit message to the followers.

By following these steps, the protocol ensures that a state change is properly stored before it is committed. It also guarantees that transactions are ordered, consistent, and durable, despite possible crash faults.

- **Scalability** in ZooKeeper systems can be achieved through the use of observer servers and dynamic scaling. To recap, observers are servers that replicate the state of ZooKeeper. They are used to scale the system without impacting the performance of state-changing requests because they do not participate in the voting process. Dynamic scaling, a second relevant feature, can be used to conveniently add new servers to the ZooKeeper ensemble while the system is online.

- A **client-push/server-pull** communication model can also be achieved using ZooKeeper. This is because Clients can submit either read or state-changing requests. A ZooKeeper server would push a message to a Client only if it has registered for notifications on state changes in a znode by setting a watch. Since this is an optional feature that is possible only through the willing participation of a Client, the required communication model holds.

- **At-least-once** message delivery semantics can be reached using ZooKeeper as long as permanent znodes with proper access rights are used for the submission of service calls. Watches set using read operations can then be used to minimise the possibility of a missed submission.

- **Recoverability** and **security** are possible because of ZooKeeper's use of transaction logs and snapshots for data persistence on local storage, and mutual authen-

tication, SSL-encrypted communication, and access control measures, respectively. For further information on these features, please refer back to Section 4.1.

Since all of the requirements of Table 4.3 can be met using ZooKeeper, the next section details the design and implementation of the queuing service using this platform.

4.4.3 Service Call Throttling

This section is concerned with using the mechanisms available to ZooKeeper to implement a queuing service. As such, it specifies the requirements and implementation details of an integrated OPC UA and ZooKeeper solution for mediating OPC UA service calls.

Requirements

There are several requirements for the operation of the queue. First, the system should allow OPC UA Servers to register and initialise a queue. OPC UA Clients and Servers must also be able to assign and retrieve tasks to and from their respective queues. The order in which tasks are processed may be important. The system should therefore give sufficient support for the ordered execution of these SCs. The system should also have mechanisms allowing OPC UA Clients to detect when a Server has crashed or disconnected. This may be useful, for example, for cases when an OPC UA Client can communicate directly with an OPC UA Server, but the Server is unable to communicate with and retrieve assigned tasks from ZooKeeper.

OPC UA Clients may also need to circumvent the queue when needed. For example, in a safety-critical environment, certain tasks may need to be processed by a server immediately, regardless of the length of the queue. Thus, it may be necessary to keep an OPC UA-native back-channel open to allow Clients to invoke service calls on Servers directly.

Lastly, a typical pattern for MOM based message queuing involves publishing the response message on the same platform [194]. This is not necessary for the queueing service. This is because the service's purpose is to shield OPC UA Servers from excessive concurrent service calls by Clients, and not vice versa. Hence, there is no immediate need to queue Server responses as well.

With these demands in place, the next subsection will describe the implementation of a ZooKeeper queuing service. This includes the data structure employed and the expected communication flow between an OPC UA Client, ZooKeeper server, and OPC UA Server.

Implementation

A prototypical implementation of the queueing service is built using the open source C99 implementation of OPC UA, open62541 [201], and the ZooKeeper library [193]. This subsection presents the data structure and the communication flow used for the

implementation, which itself is also open source[8]. The overall system architecture can be seen in Fig. 4.14.

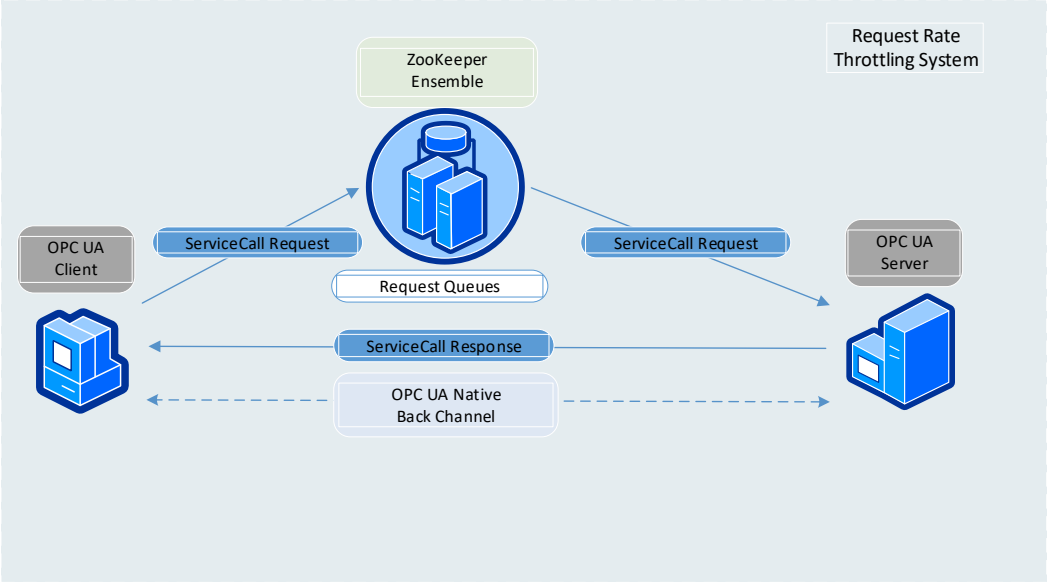

Figure 4.14: The integrated system architecture.

The data structure used by the queueing service builds upon previous work employed in [208] where ZooKeeper is used for the coordination of redundant OPC UA Servers. The main feature adopted is the use of a GUID for each set of redundant OPC UA Servers. In the queueing service, this allows clients to set watches in a more specific manner that is expected to reduce the overall frequency of notifications received. The resulting data structure for service registration, queue management, and crash detection would be as shown in Table 4.5 and Fig. 4.15.

The root path for the queuing service on ZooKeeper is the */Servers* znode. Each set of redundant OPC UA Servers is assigned its own unique path under the */Servers* znode using its GUID. The resulting path is therefore */Servers/GroupGUID*. Any active server in the redundancy set then registers itself under the */Servers/GroupGUID/Active* znode using its ServerUri or ServerId. A ServerUri or ServerId is used to identify a specific server out of a redundancy set in the case of non-transparent or transparent server redundancy, respectively.

Once an OPC UA Server registers itself as an active server, it initialises a queue using its ServerUri/ServerId under the *Servers/{GroupGUID}/Queue/* znode. OPC UA Clients can submit tasks to a Server's queue as permanent and sequential znodes.

[8]https://github.com/AGIsmail/UaRateThrottling

ZooKeeper Path	Type of znode	Explanation
/Servers	Persistent	The root folder for zkUA Servers.
/Servers/{GroupGUID}	Persistent	A GUID unique to each OPC UA Server redundancy set.
/Servers/{GroupGUID}/Active	Persistent	Path to registered zkUA Servers that are in a functional state and connected to a downstream device.
/Servers/{GroupGUID}/Active/{ServerUri}	Ephemeral	Each active server registers itself using an ephemeral znode.
/Servers/{GroupGUID}/Queue/	Persistent	Path to the queues of registered zkUA Servers that are in a functional state and connected to a downstream device.
/Servers/{GroupGUID}/Queue/{ServerUri}	Persistent	Every OPC UA Server creates a persistent znode named after its server URI under the Queue path to accept task assignments.
/Servers/{GroupGUID}/Queue/{ServerUri}/Tasks-#	Persistent-Sequential	Clients assign tasks to a Server by creating a znode with the service call and the needed arguments under the correct Server's Tasks path. Each task is a sequential znode for the synchronous execution of calls.

Table 4.5: The ZooKeeper data model for OPC UA service call queuing.

Thus, after the queue has been initialised, the Server sets a watch on its tasks queue using a read operation to monitor for new tasks. If tasks have already been queued between the time that the queue is initialised and read, they are retrieved, processed, and deleted from ZooKeeper. As a watch is set on the queue by the Server, any future tasks added by Clients trigger a single notification with the creation of a task znode. No further notifications are sent until the Server retrieves the task list and re-sets the watch. A sequence diagram demonstrating this communication flow between an OPC UA Client, ZooKeeper Server, and OPC UA Server is shown in Fig. 4.16.

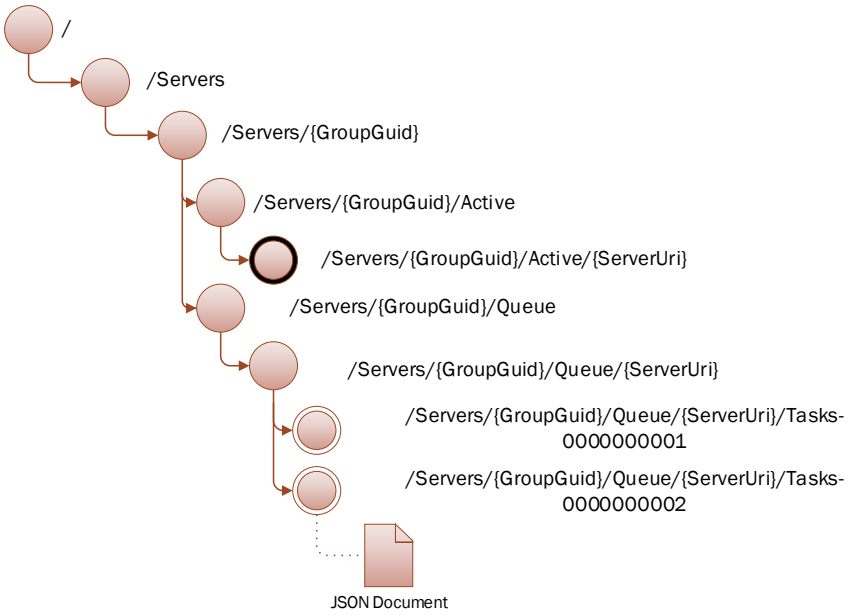

Figure 4.15: The hierarchical ZooKeeper data model for OPC UA service call queuing.

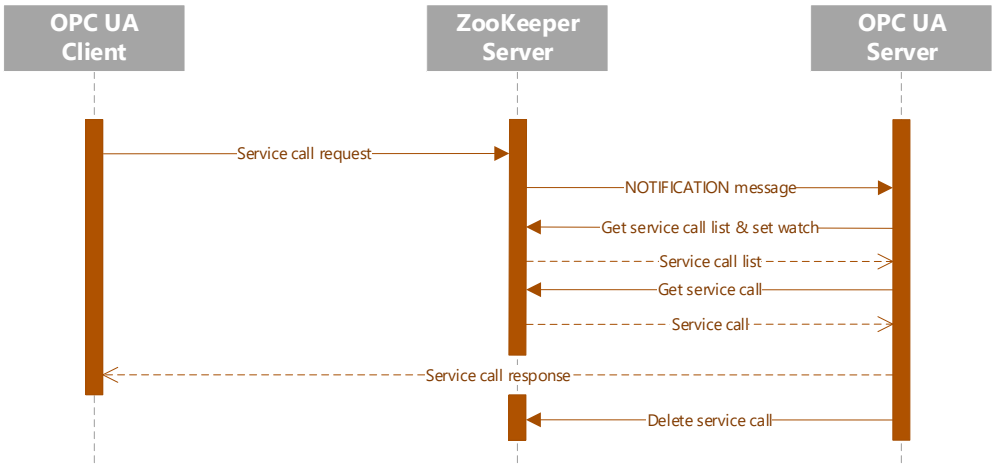

Figure 4.16: Changes to the client-server service call communication flow.

4.4.4 Discussion

The previous section presented the data structure and process flow for the queuing and processing of OPC UA service calls. An immediate concern revolves around the possibility of service calls being processed more than once or not at all. The former may occur if the queue is retrieved more than once before the OPC UA Server has had a chance to process and delete all of the tasks retrieved in a previous run from ZooKeeper. The prototype currently prevents this from occurring. Once a notification is triggered, the watcher function synchronously retrieves the task list, re-sets the watch, and processes each task before returning. The watcher function is not triggered again until it has completed, and therefore deleted the tasks on ZooKeeper and returned. This is the desired order of events as it is a requirement of Subsection 4.4.3 that tasks have to be executed in order.

If, however, the ordered execution of service calls is not important, then the watcher function may asynchronously execute its tasks and return early. This may allow tasks to be executed more than once. Recall, however, that tasks are submitted to ZooKeeper as sequential znodes. Preventing the multiple executions of a service call could therefore be as simple as having each thread processing a task list only execute znodes with an ID higher than the last ID of its predecessor and lower than its own last task's ID. Alternatively, an internal queue, e.g., using a hashtable, could be used to store the retrieved tasks and their status. Multiple threads can then contend over the available tasks and use locks to relieve the possibility of duplicate executions.

Another issue to be addressed is the possibility that service calls time out while in the queue. However, the OPC UA specifications do not currently specify a value for message time-outs. This value is purposely left open for developers to set. Needless time-outs can therefore be avoided through the selection of a reasonable value that reflects the context of execution.

The Clients' ability to circumvent the queue should be designed with the utmost care. The use of OPC UA's native communication flow for service calls effectively pushes requests to the front of the queue. This may overload a server and cause it to enter a degraded or failed state. It may also alter the states previously used to submit the currently queued requests and may result in their unsafe execution. Out-of-queue service calls should therefore be done with caution.

Finally, in [31], it is indicated that distributed queues is one of the least used applications for ZooKeeper. The explanation offered is that consensus may have a detrimental effect on the performance of large queues. Apache Curator's documentation [209] recommends against the use of ZooKeeper for queues because of an anecdotal report in [210]. The report mentions ZooKeeper's transport limitations, slow start up times and other complexities introduced by having large queues. However, the points listed are all self-admittedly born of casual observation in a global and multi-tenant architecture. Thus, these remarks are not based on rigorous scientific analysis, are not quantified, and are asserted by only a single source. The problems mentioned are also stated as being the result of "[developers abusing] the queues" [210]. Given the difference in context between the environment used in [210] (a global multi-tenant architecture for Netflix)

and a manufacturing environment, results may differ. This work does not currently pursue an evaluation to discern these limits which may include the size of the ZooKeeper ensemble, number of active OPC UA clients and servers, znode size and distribution per size, queue lengths, ZooKeeper settings (e.g., forceSync), and hardware performance (e.g., storage I/O) [211]. This, in fact, is a problem that may garner different results between deployments. Instead, the prototype developed for this work is open-sourced and it should be possible to design appropriate architectures, quantify normal limits of operation, and address contingencies for the scalable execution of the service and dependent systems once feedback is available from early adopters.

Future work may also investigate the use of load shedding for server overloads as was mentioned in the introduction of this chapter. A comparison of server-side load-shedding and task queueing on ZooKeeper would be immensely useful. A possible outcome of this analysis may shed light on the engineering cost involved in determining which service calls can be dropped versus the cost of ensuring safe out-of-queue service calls.

4.5 Conclusion

In this chapter, the coordination requirements of OPC UA Server redundancy were quantified and shown to be realisable using the ZooKeeper service. An explanation of the overall architecture, data model, and components of the integrated OPC UA and ZooKeeper system was given. This includes the appropriate consideration of a solution for the migration of existing OPC UA systems. An example implementation based on the open source ZooKeeper and open62541 libraries was described.

While real-world deployments would still require careful design to ensure that sufficient resources are present for safe operation, the resulting system should be capable of providing a reliable framework for OPC UA Server redundancy. Through this system, redundant servers may achieve the required goals of synchronisation and replication, failure detection, failover initiation, and resource fencing. The extensibility of the data model given should present opportunities to accommodate further synchronisation requirements than those shown in this chapter. It is expected that future iterations of this system address more complex features of OPC UA, such as Method Node replication, and include more technologies from the IT domain to address other open questions in the standard.

This chapter also addressed the possible vulnerability of OPC UA Servers to resource exhaustion due to high rates of concurrent service calls by OPC UA Clients. A queuing service for service call rate throttling was identified as a possible solution for mitigating server overload. The requirements for this service were determined and shown to be achievable using Apache ZooKeeper. A data structure and queuing protocol is designed and demonstrated using a prototypical implementation based on the same open62541 and ZooKeeper libraries.

While certain facets of the design necessitate care in use and administration, the system meets all of the demands determined for the reliable queuing and execution of service calls. Future work for this service is expected to focus on comparisons with alter-

native solutions, as well as the determination of appropriate deployment architectures and best-practices that account for identified contingencies.

.

CHAPTER 5

Conclusion & Outlook

As discussed in Chapter 1, manufacturing strategies are typically used by an enterprise to give assurances that its decisions match with its requirements and vision. This dissertation applied itself within the context of the smart manufacturing approach for the design and implementation of infrastructural systems for resilient M2M communication in distributed CPPS. Thus, it investigated the application of SOAs, M2M communication middleware systems, and overlay networking solutions to improve the agility, resilience, and interoperability of enterprise infrastructure. This chapter provides a summary on the main contributions of this dissertation and provides a discussion on possible directions for future work.

5.1 Summary of Contributions

5.1.1 Service Oriented Architectures

Manufacturing systems require infrastructural agility, interoperability, and flexibility to meet the goals of lean and agile manufacturing. Existing systems, however, continue to have complex and heterogeneous technical infrastructure that impede the progress of achieving these properties. This is the first problem statement identified in Section 1.3 - [**PS-1**]. For this purpose, Chapter 2 investigates the hypothesis that SOAs "provide a suitable pathway for the pursuit of agile characteristics in modern enterprises" [**H-1**]. Thus, it explains the properties of SOA and demonstrates how they may be used in CPPS to counteract the systemic complexity that hinders manufacturing processes.

Yet, a survey on service-based manufacturing systems carried out by Kevin Nagorny *et al.* in [38] demonstrates that a large number of SO RAs exists for CPPS. Herein lies the second problem statement of this dissertation which states that "the proliferation of SOA in the manufacturing domain complicates the process of selecting an appropriate RA for the development of SO manufacturing infrastructure" [**PS-2**]. To moderate this complexity, Chapter 2 also performs a state-of-the-art analysis on the preliminary

SO RAs of five major European projects: IMC-AESOP, PLANTCockpit, IoT@Work, eScop, and Arrowhead framework. The five RAs are analysed for their practicality by determining the simplicity of transforming them into concrete implementations. The architectures, technologies, and communication stacks used in their realisations are presented. Results attained through the application of an analysis framework developed in [32] show that the SO RAs are often under or over specified and, on occasion, are missing critical elements from their specifications implying that extra effort may be required for their implementation. By extrapolating the results of [32] to this survey, the review concludes that the SO RAs presented are vulnerable to low adoption rates and criticisms from stakeholders.

Thus, Chapter 2 pursues the second hypothesis of this dissertation that the "mature, standardised, well-adopted, and well-supported" OPC UA specifications may provide a SO solution that is well-suited for "the integration of M2M communication infrastructure" [**H-2**]. This is confirmed by the application of the same methods used to evaluate the SO RAs of the five major European projects. Based on this conclusion, OPC UA is adopted as the underlying technology for the development of SO manufacturing systems for the remainder of the dissertation.

5.1.2 Cooperative P2P Overlay Networks

Following a bottom-up approach, the first task to tackle in developing a resilient SO system involves addressing the need for "flexible, scalable, and failure-resistant transport protocols for a dynamic system of services. This is in contrast to current technologies which often limit these properties" [**PS-3**].

The third hypothesis of Section 1.3 describes P2P networks as cooperative systems as a field with possibly suitable solutions to this problem, albeit, "deviations from their typical nature will be necessary to adapt them to the manufacturing domain" [**H-3**].

Consequently, Chapter 3 discerns the properties of these networks and develops a resilient transport layer for middleware systems based on their principles. Due to these properties, the resulting protocol may allow participating manufacturing infrastructure to function as independent clusters that may undergo runtime reconfiguration for creating expanded systems, inter-system traffic engineering, and inter-system content sharing. This allows middleware to dynamically adapt to changes in the requirements and policies of the network architecture in a manufacturing enterprise and introduces a resilience to the transport layer against node and link failures. A typical P2P protocol, Chimera, is used to demonstrate the conversion of a traditional protocol into a cooperative systems one. A prototypical implementation is developed and evaluated via virtual deployments on a Xen Project server and 32-bit embedded devices. Empirical results demonstrated that the discovery mechanisms typically employed by P2P technologies are unfit for manufacturing infrastructure due to high traffic loads and slow discovery times. Thus, a SO approach is followed to separate the discovery, management, and networking elements. Service discovery is then implemented using the low traffic zero-conf mDNS protocol. The prototypical evaluation using the Avahi mDNS and modified cooperative networks Chimera libraries demonstrated full-network discovery in less than 3 seconds.

An added feature of the resulting system is its agnosticism towards the used middleware technology. Thus, it is compatible with both OPC UA and non-OPC UA applications in manufacturing.

5.1.3 OPC UA-Based M2M Communication Middleware Systems

With the transport layer addressed, the remainder of the dissertation focused on enhancing the OPC UA based application layer. The fourth problem statement of this dissertation states that "OPC UA based distributed systems have coordination needs for the safe operation of redundant servers that would require an extensive investment on the part of developers" [**PS-4**].

A review of the SO OPC UA M2M communication specifications highlights the need for address space synchronisation, failure detection, and resource fencing. The demands of these three coordination functions are discerned and, in accordance with the fourth hypothesis [**H-4**], are shown to be realisable using Apache ZooKeeper as a coordination platform. Therefore, a system architecture, data model, and component descriptions are designed and detailed. An open source evaluation is implemented using the open62541 and Apache ZooKeeper libraries showing that the system is capable of realising the required goals of runtime address space synchronisation, failure detection, and barrier synchronisation.

The fifth problem statement concerns itself with the vulnerability of OPC UA servers to resource exhaustion due to the client-server communication mechanisms in place [**PS-5**]. This vulnerability is found to be due to clients' unhindered ability to transmit SCs to a server. This means that a server may potentially receive an overload of requests causing it to fail or operate in a degraded state as it tries to process them. A SC queuing service is proposed as a solution to throttle the rate of concurrently processed requests by a server [**H-5**]. The requirements of the service are discerned and Apache ZooKeeper is shown to be a viable platform for its development. Similar to the coordination service, the architecture, data model, and service components are detailed. An open source implementation is carried out using the open62541 and Apache ZooKeeper libraries. A back channel for circumventing the queue is also integrated in the system for increased operational safety. The overall service is shown to be able to meet the requirements of a queuing service for throttling the rate of SCs to an OPC UA server.

5.1.4 Discussion

In sum, this dissertation focused on delivering an integrated system of service overlays that provide a runtime environment for the deployment and execution of resilient services, as shown in Fig. 5.1. The system is based on SOA, P2P networks as cooperative systems, and OPC UA to allow for the development of distributed, dynamic, and robust services for CPPS. While the proposed work is developed in the context of the classical automation pyramid based on PERA and associated standards, the contributions of this dissertation may be specified with respect to the functional hierarchy of the three-dimensional RAMI 4.0 [212]. This seminal architecture, shown in Fig. 5.2, is a three-

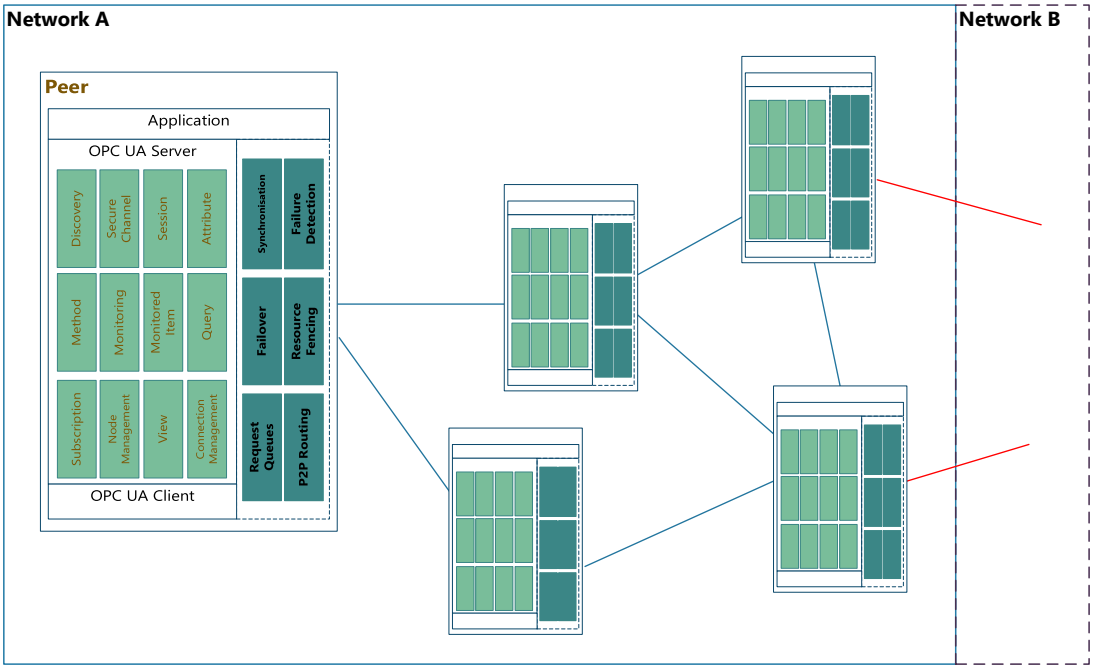

Figure 5.1: The main building blocks of the service overlays for resilient CPPS.

dimensional coordinate system of elements identified for the fourth industrial revolution, also known as Industrie 4.0 (I4.0). The three axes represent the hierarchy levels (right horizontal), life cycle and value streams (left horizontal), and layers (vertical) in an I4.0 environment [213].

The hierarchy levels expand upon those stated in the IEC 62264 (ISA-95) standard for enterprise integration by including the new 'Product' and 'Connected World' levels. The former represents the work piece and the latter the IoT. Thus, given the dissertation's focus on enhancing the features of enterprise infrastructure, the main contributions are concerned with the 'Field Device', 'Control Device', 'Station', 'Work Centres', and 'Enterprise' levels. Given the nature of the technologies adopted for the dissertation, it should also be possible to extend the designed system to include the 'Product' and 'Connected World' levels given appropriate effort [213]. This is discussed further in the coming section.

The life cycle and value streams axis in RAMI 4.0 predominantly applies to the manufactured product. Therefore, it is composed of two overarching elements: the type and the instance. To quote, "a type becomes an instance when design and prototyping have been completed and the actual product is being manufactured" [213]. Thus, this axis may be accurately labelled as the product life cycle. The accomplishments of this dissertation are primarily concerned with the production system life cycle as they address concepts for the operation of systems that are the means through which resources are

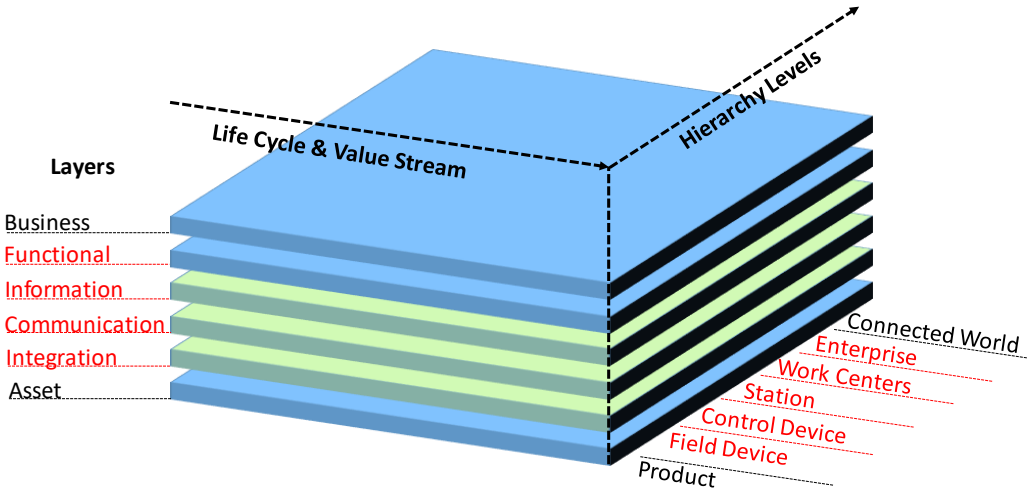

Figure 5.2: Translating the contributions to the RAMI 4.0. Addressed layers and hierarchical levels are highlighted in red.

converted into manufactured products. However, as pointed out in the previous paragraph, extensions are possible and may be applied here to incorporate aspects of the life cycle and value streams axis of RAMI 4.0 as well.

Finally, the vertical axis of Fig. 5.2 decomposes the properties of a machine into six definitive layers. Of these, the system incorporates aspects of everything from the Functional to the Integration layer. For example:

- Functional: the inclusion of remote access and horizontal integration.
- Information: the "provision of structured data via service interfaces" [213].
- Communication: standardised communication and data formats and the provisioning of services for controlling the integration layer.
- Integration: provisioning processable information from assets and event generation.

Hence, it can be concluded that the degree of congruence between the features of the envisioned system and the governing concepts of a pioneer R&D project such as RAMI 4.0 serves as evidence to the integrity of the delivered contributions. The system introduces numerous desirable properties to CPPS with room to extend the vision of the system beyond its current boundaries. This, alongside other possible enhancements to the system, is the topic of discussion for the coming section.

5.2 Future Work

Extending to other environments

As of yet, the contributions of this dissertation are focused on improving the state of infrastructure local to the manufacturing enterprise. These CPPS are predominantly concerned with the safe collection of data and execution of control over local resources. The nature of the manufacturing ecosystem has, however, changed to apply significant importance to data outside of the production system as well for decision making processes [14]. This includes data from product development and supply chain life cycles. Thus, there is a need for the integration of systems and information flows from these currently isolated dimensions. The applied technologies and strategies can be ported over to aid in this process of integration to provide the same advantages of agility, flexibility, and resilience. However, a different context, again, implies different use cases, requirements, and thus, technologies and implementations. The design of SO M2M communication platforms for the automated acquisition and integration of data from these life cycles is an expensive task that warrants investigation in the immediate future. While this should henceforth serve as the governing strategy for the development of the platform, the currently existing technical features of the system should also receive appropriate attention. Several specific points for their development are discussed in the coming subsections.

Enhancements to OPC UA Middleware

As applied to the coordination service for redundant OPC UA servers, an immediate enhancement to the presented system would be a solution to Method node replication. The evaluation of a SO system of services for the various possible methods is suggested as a viable solution in Chapter 4. The requirements and resulting architecture and implementation details to allow for the dependable operation of OPC UA applications is a considerable challenge. Given the foreseen number of services that may run concurrently on a single device, this may result in demands for lightweight processing and communication technologies with guarantees for deterministic execution and bounded response times. Supporting infrastructure for versioning, management, and monitoring, amongst others, may also be necessary for their coordination.

A second possibility for future work highlighted in Chapter 4 involves the investigation of load shedding as a solution for resource-strained OPC UA servers. As previously pointed out, this is based on the premise that rejecting a SC consumes less resources than processing it. Given the close coupling of computational and physical resources in a CPPS, arbitrarily rejecting a SC may have undesirable physical consequences to the manufacturing system or product. Thus, mechanisms are required to establish a clear understanding of the result of denying a specific SC in its context. The available options for such a system need to be investigated to discern the engineering costs and runtime overheads associated with candidate systems.

Enhancements to the Transport Layer

An immediate task available for enhancing the discovery feature of the cooperative P2P networks transport layer was presented in Chapter 3. This relates to the complete abandonment of unicast-based network heartbeat messages in favour of a mechanism based on the tracking of service advertisement removal. However, the vulnerability of advertisement removal announcements to immediate link and node failures on the service host must first be addressed. Chapter 3 has proposed and discussed the use of modules modelled on the concept of IPMI subsystems as a solution. An investigation of an algorithmic software-based approach is also warranted given the cost and engineering factor involved in developing such a subsystem.

An investigation into improving the mDNS reflector service of the P2P network is also discussed in Chapter 3. This may, for example, involve the aggregation of service advertisements at reflector nodes and their announcement in foreign subnets as resources local to the reflectors. Since this may cause a concentration of traffic on reflector nodes, strategies for the distribution of routing load amongst the gateways may be required for this architecture. The second proposed modification is the caching of service announcements at reflector nodes to reduce the load of propagating service queries. This, in turn, would require a careful study of the repercussions of caching on the passive failure discovery features of mDNS.

A final point worth investigating is in addressing the transport layer's limitation to non-RT communication tasks. Given the importance of RT systems in manufacturing operations, an apparent next step would be to consider the introduction of this feature to the cooperative P2P transport layer. Previous work in [214, 215] demonstrates the viability of RT communication in P2P networks through a hard RT implementation of a Kademlia network. However, as of yet, the engineering of a RT cooperative systems protocol has not been examined in an academic context. Doing so may, in theory, allow the same advantages of resiliency and flexibility to be extended to field level equipment. However, the standards, technologies, and requirements of field level systems differ significantly from the upper layers of manufacturing infrastructure. This, in turn, implies different demands for the cooperative systems protocol, possibly creating a different breed of networks.

Acronyms

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks. 30, 40, 41

ACL Access Control Lists. 118, 130

AMQP Advanced Message Queuing Protocol. 30, 38, 39

API Application Programming Interface. 11, 22, 32, 34

ARB Access Request Broker. 22

AS Authorisation Service. 22

AutomationML Automation Markup Language. vii

BASH Bourne Again Shell. 96, 97, 104

CAN Controller Area Network. 30, 40, 41

CAN Content Addressable Network. 63, 68, 69, 81

CBAC Capability-Based Access Control. 30, 41, 42

CBOR Concise Binary Object Representation. 37

CEP Complex Event Processing. 20, 26

CHK Content-Hash Key. 74

CoAP Constrained Application Protocol. 30–32, 38, 39, 41

CoRE Constrained RESTful Environments. 30, 32, 34, 35

CPPS Cyber-Physical Production Systems. vii, ix, 1, 11, 13, 17, 111, 132, 141, 143–146

CSI Communication Service Interface. 22

DC Doctoral College. vii

DCS Digital Control System. 20

DER Distinguished Encoding Rules. 47, 52

DHCP Dynamic Host Configuration Protocol. 30, 40, 41

DHT Distributed Hash Tables. xiii, 61, 62, 71, 78, 80–82, 84, 86

DMZ Demilitarised Zone. 5

DNS Domain Name System. 27, 30, 31, 33, 34, 41, 42, 77, 93, 95, 101

DNS TSIG Domain Name System Transaction SIGnature. 30, 41, 42

DNS-SD Domain Name System Service Discovery. 30, 31, 33, 34, 40, 103, 109

DNSSEC Domain Name System Security Extensions. 30, 41, 42

DoS Denial of Service. 107

DOT Direct Overlay Table. 85

DPWS Device Profile for Web Services. 30–32, 34, 38–41, 43

DS Directory Service. 22, 32, 36, 41

DTD Document Type Definition. 37

DTLS Datagram Transport Layer Security. 30, 38, 41, 42

EDBT Extending Database Technology. vii

ENS Event Notification Service. 22, 39

EPL Expected Path Lengths. 80

ERP Enterprise. 21, 22

eScop Embedded systems Service-based Control for Open manufacturing and Process automation. 18, 20, 27, 28, 30–34, 36, 38, 41, 43, 45, 46, 142

eScopRTU Embedded systems Service-based Control for Open manufacturing and Process automation Remote Terminal Unit. 34

EU European Union. ix, 2, 13

EXI Efficient XML Interchange. 30, 33, 36, 37, 41, 109

FB Function Block. 32

FIFO First-In-First-Out. 114

FMC Fundamental Modelling Concepts. 24, 46

FOGP Full OGP peers. 86

GDB GNU Debugger. 96

GDS GlobalDiscoveryServer. 47, 48

GPRS General Packet Radio Service. 30, 40, 41

GSM Global System for Mobile Communications. 30, 40, 41

GUID Globally Unique Identifier. 122, 123, 125, 136, 137

HMI Human-Machine Interface. 20, 23, 24, 26, 27, 36

HoHA Horizontal Hierarchical. 59, 62

HTL Hops-to-Live. 73

HTTP Hypertext Transfer Protocol. 28, 30–33, 35, 36, 38–41, 43, 47, 51–53

HTTPS HTTP over SSL/TLS. 30, 38–40, 47, 49, 51, 52

i3 Internet Indirection Infrastructure. 102

I4.0 Industrie 4.0. 144

IA Information Assurance. 29

IEEE Institute of Electrical and Electronics Engineers. 17, 30, 40–42, 57, 111

II Information Infrastructure. 29

IMC-AESOP ArchitecturE for Service-Oriented Process - Monitoring and Control. xiv, 18, 20, 24, 25, 27, 30–41, 43, 45, 46, 108, 142

INT Interface Layer. 27

IOOM Inter-Overlay Optimisation Manager. 89

IoT Internet of Things. 1, 20, 144

IoT@Work Internet of Things at Work. xiv, 18, 20–22, 30–34, 36–42, 45, 46, 108, 142

IPMI Intelligent Platform Management Interface. 107, 147

IPv4 Internet Protocol version 4. 30, 40, 41

IPv6 Internet Protocol version 6. 30, 40, 41, 43, 100

IT Information Technology. 1, 5, 8, 140

JMS Java Message Service. 30, 38, 39, 41, 42

JSON JavaScript Object Notation. 30–34, 36–38

KSK Keyword-Signed Key. 74

LDAP Lightweight Directory Access Protocol. 30, 41, 42

LDS LocalDiscoveryServer. 47, 48

LDS-ME LocalDiscoveryServer with MulticastExtension. 47, 48

LOGP Lightweight OGP peers. 86

LTM Location-aware Topology Mechanism. 89

M2M Machine-to-Machine. ix, 8, 11–13, 38, 141–143, 146

MBOM Mesh-Based Overlay Manager. 89

mDNS Multicast Domain Name System. 30, 31, 33, 101, 103, 104, 108, 109, 142, 147

MES Manufacturing Execution System. 5, 22

MIB Management Information Base. 30, 34

MOM Message Oriented Middleware. 38, 39, 135

MPI Message Passing Interface. 30, 41, 42

MQTT Message Queue Telemetry Transport. 30, 38, 39

MRR Maximum-Replication-Rate. 84

MRT Message Routing Table. 85

MSO Manufacturing Systems Ontology. 28, 36

NAC Network Access Control. 20, 42

NFC Near-Field Communication. 30, 32, 40–42

NIST National Institute of Standards and Technology. 112

NTP Network Time Protocol. 30, 40, 41, 51

NTR Non-Transparent Redundancy. 119, 120

OAS OpenAPI Specification. 34

OGP Overlay Gateway Protocol. 86

OPC Open Platform Communications. 54

OPC UA Open Platform Communications Unified Architecture. ix, 8, 11–14, 30–41, 46–55, 58, 110–113, 118–125, 128–133, 135–140, 142, 143, 146

ORL Orchestration Layer. 27, 28, 43

P-PSO Politecnico di Milano Production Systems Ontology. 36

P2P Peer-to-Peer. ix, 11, 13, 58, 59, 62, 63, 65–67, 74, 78–81, 87, 88, 90–94, 102–104, 109, 142, 143, 147

PDP Policy Decision Point. 22

PERA Purdue Enterprise Reference Architecture. 5, 143

PHL Physical Layer. 27, 28, 43

PKI Public Key Infrastructure. 30, 41, 42

PLANTCockpit Production Logistics and Sustainability Cockpit. 18, 20, 21, 23, 24, 30–34, 36–42, 45, 46, 142

PTP Precision Time Protocol. 30, 40, 41

QoS Quality of Service. 20, 22, 33

QR Quick Response. 30, 32, 40, 41

RA Reference Architecture. ix, 8, 13, 18, 20, 27, 31, 43–46, 53, 54, 141, 142

RAMI 4.0 Reference Architecture Model Industrie 4.0. 15, 143–145

RBAC Role-Based Access Control. 30, 41, 43

RD Resource Directory. 32

RDF Resource Description Framework. 36

REST Representational State Transfer. 30–34, 36, 37, 42

RPC Remote Procedure Calls. 14, 38, 112

RPL Representation Layer. 27–29, 43

RT Real Time. 5, 39, 41, 109, 110, 147

RTU Remote Terminal Unit. 34

SAML Security Assertion Markup Language. 30, 36, 37, 41, 42

SC service call. ix, 13, 14, 135, 143, 146

SCADA Supervisory Control and Data Acquisition. 14, 20, 111

SDK Software Development Kit. 40, 131

SenML Sensor Markup Language. 30, 33–35, 41

SEP Slice Enforcement Point. 22

SM Systems Management. 29

SNMP Simple Network Management Protocol. 30, 34, 36, 40, 41

SO Service Oriented. ix, xii, 8, 10–13, 17, 18, 20, 42, 43, 46, 54, 58, 108, 109, 111, 141–143, 146

SOA Service Oriented Architecture. ix, 3, 4, 8, 12, 13, 17, 43, 46, 103, 107, 109, 110, 141, 143

SOAP Simple Object Access Protocol. 31, 37–40, 47, 51–53, 109

SOM Single Overlay Manager. 89

SPARQL SPARQL Protocol and RDF Query Language. 28, 36

SPoF Single Point of Failure. 8, 60, 63

SSH Secure Shell. 30, 41, 42

SSK Signed-Subspace Key. 74

SSL Secure Sockets Layer. 118, 130, 133, 134

SSO Single Sign-On. 30, 41, 42

STL Structured Text Language. 34

STP Spanning Tree Protocol. 30, 40, 41

SysML Systems Modelling Language. 46

TCP Transmission Control Protocol. 38–41, 47, 49, 51, 68, 93, 114

ThingML Thing Markup Language. 30, 34, 36

TLS Transport Layer Security. 30, 38, 41–43

TR Transparent Redundancy. 118–120

TTL Time-to-Live. 73, 79, 80, 83, 84, 86, 100

UDP User Datagram Protocol. 31, 38, 39, 41, 42, 68, 93, 109

UI User Interface. 24

UML Unified Modelling Language. 46, 54, 124–128

UMTS Universal Mobile Telecommunications System. 30, 40, 41

URI Uniform Resource Identifier. 35, 38, 48, 118, 125, 136

URL Uniform Resource Locator. 32, 75, 118

UWB Ultra-Wide Bandwidth. 30, 40

VeHA Vertical Hierarchical. 59, 62, 63, 76, 80

VIS Visualisation Layer. 27–29, 43

VLAN Virtual Local Area Network. 41

VM Virtual Machine. 96, 104, 109

WADL Web Application Description Language. 30, 33, 34

WS Web Services. 28, 40, 51, 53, 109

WS-Discovery Web Services Dynamic Discovery. 30–34, 39, 108, 109

WSDL Web Services Description Language. 30, 33, 34

WSN Wireless Sensor Networks. 32, 33, 43

XACML eXtensible Access Control Markup Language. 30, 36, 37, 41, 42

XML eXtensible Markup Language. 30, 31, 33–42, 51, 109

XMPP Extensible Messaging and Presence Protocol. 30, 38, 39, 42

XOP/MTOM XML-binary Optimized Packaging and Message Transmission Optimisation Mechanism. 30, 36, 37

XOR Exclusive OR. 61, 64, 74

XSD XML Schema Definition. 37

YAML YAML Ain't Markup Language. 30, 34, 36–38

zkUA Zookeeper integrated OPC UA. 121–131, 136

zxid ZooKeeper Transaction ID. 117, 129

Bibliography

[1] *Industrial Internet of Things: Unleashing the Potential of Connected Products and Services*. Industry Agenda Report. World Economic Forum (in collaboration with Accenture), Jan. 2015.

[2] *That 'Internet of Things' Thing*. June 22, 2009. URL: http://www.rfidjournal.com/articles/view?4986.

[3] *Current Members | Industrial Internet Consortium*. URL: http://www.iiconsortium.org/members.htm.

[4] *Industrie 4.0*. URL: https://www.zvei.org/themen/industrie-40/.

[5] J. Mitchell. "The intelligent manufacturing systems initiative". In: *Proceedings of 15th IEEE/CHMT International Electronic Manufacturing Technology Symposium*. Oct. 1993, pp. 427–431. DOI: 10.1109/IEMT.1993.398170.

[6] *Production of the Future - the Programme | FFG*. URL: https://www.ffg.at/en/production-future-programme.

[7] *Manufacturing USA - the National Network for Manufacturing Innovation | Manufacturing.gov*. URL: https://www.manufacturing.gov/nnmi/.

[8] *Smart industry - a strategy for new industrialisation for Sweden*. Apr. 20, 2016. URL: http://www.government.se/contentassets/3be3b6421c034b038dae4a7ad75f2f54/nist_statsformat_160420_eng_webb.pdf.

[9] *About SMLC | Smart Manufacturing Leadership Coalition*. URL: https://smartmanufacturingcoalition.org/about.

[10] H. Corrêa. "Agile Manufacturing as the 21st Century Strategy for Improving Manufacturing Competitiveness". In: *Agile Manufacturing: The 21st Century Competitive Strategy*. Ed. by A. Gunasekaran. Oxford: Elsevier Science Ltd, 2001, pp. 3–23.

[11] E. Puik et al. "Assessment of reconfiguration schemes for Reconfigurable Manufacturing Systems based on resources and lead time". en. In: *Robotics and Computer-Integrated Manufacturing* 43 (Feb. 2017), pp. 30–38. DOI: 10.1016/j.rcim.2015.12.011.

[12] F. Bauhoff et al. *Roadmaps Executive Summary*. IMS2020, July 15, 2010.

[13] G. Chryssolouris et al. “Digital manufacturing: History, perspectives, and outlook”. en. In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 223.5 (May 2009), pp. 451–462. DOI: 10.1243/09544054JEM1241.

[14] Y. Lu, K. Morris, and S. Frechette. *Current Standards Landscape for Smart Manufacturing Systems*. Tech. rep. NIST IR 8107. DOI: 10.6028/NIST.IR.8107. National Institute of Standards and Technology, Feb. 2016.

[15] S. Iarovyi et al. “From artificial cognitive systems and open architectures to cognitive manufacturing systems”. In: *Proceedings of the 13th International Conference on Industrial Informatics (INDIN)*. IEEE, 2015, pp. 1225–1232.

[16] A. Giret and V. Botti. “Holons and agents”. In: *Journal of Intelligent Manufacturing* 15.5 (Oct. 2004), pp. 645–659. DOI: 10.1023/B:JIMS.0000037714.56201.a3.

[17] G. Putnik et al. “Scalability in manufacturing systems design and operation: State-of-the-art and future developments roadmap”. en. In: *CIRP Annals - Manufacturing Technology* 62.2 (Jan. 2013), pp. 751–774. DOI: 10.1016/j.cirp.2013.05.002.

[18] A. Choudri. “The Agile Enterprise”. In: *Manufacturing Handbook of Best Practices: An Innovation, Productivity, and Quality Focus*. Ed. by J. ReVelle. CRC Press, 2001, pp. 3–23.

[19] R. Giachetti. *Design of Enterprise Systems: Theory, Architecture, and Methods*. Florida, USA: CRC Press, 2011.

[20] T. J. Williams. “The Purdue Enterprise Reference Architecture”. In: *Proceedings of the JSPE/IFIP TC5/WG5.3 Workshop on the Design of Information Infrastructure Systems for Manufacturing*. DIISM ’93. Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1993, pp. 43–64.

[21] D. He et al. “An approach to use PERA in Enterprise Modeling for industrial systems”. In: *Proceedings of the 38th Annual Conference of the IEEE Industrial Electronics Society (IECON)*. 2012, pp. 4196–4203.

[22] P. Didier et al. *Converged Plantwide Ethernet (CPwE) Design and Implementation Guide*. Cisco Systems, San Jose, California and Rockwell Automation, Milwaukee, Wisconsin. 2011.

[23] *OPC Unified Architecture for ISA-95 Common Object Model Companion Specification*. R1.00. OPC Foundation. Oct. 2013.

[24] A. Ismail and W. Kastner. “Vertical Integration in Industrial Enterprises and Distributed Middleware”. In: *International Journal of Internet Protocol Technology* 9.2/3 (2016), pp. 79–89.

[25] E. Knapp. *Industrial network security: securing critical infrastructure networks for smart grid, scada, and other industrial control systems*. 2nd edition. Waltham, MA: Elsevier, 2014.

[26] T. Sauter, S. Soucek, and M. Wollschlaeger. “Vertical Integration”. In: *Industrial Communication Systems, Second Edition*. Ed. by B. Wilamowski and J. Irwin. London: CRC Press, 2011. Chap. 13, pp. 1–12.

[27] C. Marquis. “Imprinting: Toward a Multilevel Theory”. In: *Academy of Management Annals* 7 (2013), pp. 195–245.

[28] M. Valipour et al. “A brief survey of software architecture concepts and service oriented architecture”. In: *2nd International Conference on Computer Science and Information Technology (ICCSIT)*. 2009.

[29] P. Danielis et al. “Survey on real-time communication via ethernet in industrial automation environments”. In: *Proceedings of the 19th IEEE International Conference on Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–8.

[30] F. Junqueira and B. Reed. *ZooKeeper*. English. Sebastopol, CA: O’Reilly Media, 2013.

[31] A. Ailijiang, A. Charapko, and M. Demirbas. “Consensus in the Cloud: Paxos Systems Demystified”. In: *Proceeding of the 25th IEEE International Conference on Computer Communication and Networks (ICCCN)*. 2016.

[32] S. Angelov, P. Grefen, and D. Greefhorst. “A framework for analysis and design of software reference architectures”. In: *Information and Software Technology* 54.4 (Apr. 2012), pp. 417–431.

[33] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: service-oriented architecture best practices*. The Coad series. Indianapolis, IN: Prentice Hall Professional Technical Reference, 2005.

[34] T. Erl. *SOA design patterns*. 1st ed. Prentice Hall service-oriented computing series from Thomas Erl. Upper Saddle River, NJ: Prentice Hall, 2009.

[35] T. Erl et al. *Next generation SOA: a concise introduction to service technology & service-orientation*. Pearson Education, 2014.

[36] B. J. Krämer. “Evolution of Cyber-Physical Systems: A Brief Review”. en. In: *Applied Cyber-Physical Systems*. Ed. by S. C. Suh et al. New York, NY: Springer New York, 2014. ISBN: 978-1-4614-7336-7.

[37] J. Rubio. “Service Oriented Architecture for Embedded (Avionics) Applications”. PhD thesis. Technical University of Catalonia, 2011.

[38] K. Nagorny, A. W. Colombo, and J. Barata. “A survey of service-based systems-of-systems manufacturing systems related to product life-cycle support and energy efficiency”. In: *12th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2014, pp. 582–587.

[39] D. Rotondi et al. *D1.3 - Final Framework Architecture Specification*. Tech. rep. IoT@Work Consortium, July 2013.

[40] PLANTCockpit Consortium. *D3.1 Initial Architectural Components of PLANTCockpit*. Tech. rep. Sept. 2011.

[41] A. Colombo, T. Bangemann, and S. Karnouskos. "IMC-AESOP outcomes: Paving the way to collaborative manufacturing systems". In: 12th International Conference on Industrial Informatics (INDIN), July 2014.

[42] eScop Consortium. *1st Annual Report*. Tech. rep. May 2013.

[43] F. Blomstedt et al. "The arrowhead approach for SOA application development and documentation". In: 40th Annual Conference of the IEEE Industrial Electronics Society (IECON), Oct. 2014, pp. 2631–2637.

[44] A. Colombo et al., eds. *Industrial Cloud-Based Cyber-Physical Systems*. en. Cham: Springer International Publishing, 2014. DOI: 978-3-319-05624-1.

[45] S. Iarovyi et al. "Architecture for Open, Knowledge-Driven Manufacturing Execution System". In: *IFIP International Conference on Advances in Production Management Systems (APMS)*. Vol. 460. Cham: Springer International Publishing, 2015, pp. 519–527.

[46] eScop Consortium. *eScop Architecture*. Tech. rep.

[47] P. Varga et al. *Deliverable D7.3 of WP 7*. Tech. rep. Arrowhead Consortium, June 2014.

[48] F. Blomstedt et al. *Deliverable D8.3*. Tech. rep. Arrowhead Consortium, June 2014.

[49] J. Beatty et al. *Web Services Dynamic Discovery (WS-Discovery) Version 1.1*. Organization for the Advancement of Structured Information Standards.

[50] P. Nappey et al. "Migration of a legacy plant lubrication system to SOA". In: 39th Annual Conference of the IEEE Industrial Electronics Society (IECON), Nov. 2013.

[51] B. Bony, M. Harnischfeger, and F. Jammes. "Convergence of OPC UA and DPWS with a cross-domain data model". In: *9th IEEE International Conference on Industrial Informatics (INDIN)*. 2011.

[52] J. Eliasson et al. "A SOA-based framework for integration of intelligent rock bolts with internet of things". In: *IEEE International Conference on Industrial Technology (ICIT)*. 2013.

[53] R. Kyusakov et al. "EXIP: a framework for embedded Web development". In: *ACM Transactions on the Web (TWEB)* 8.4 (2014), p. 23.

[54] S. Iarovyi, J. Garcia, and J. L. M. Lastra. "An approach for OSGi and DPWS interoperability: Bridging enterprise application with shop-floor". In: *11th International Conference on Industrial Informatics (INDIN)*. 2013.

[55] A. Dennert et al. "Advanced Concepts for Flexible Data Integration in Heterogeneous Production Environments". en. In: *IFAC Proceedings Volumes* 46.7 (May 2013), pp. 348–353.

[56] S. Iarovyi et al. “CyberPhysical Systems for Open-Knowledge-Driven Manufacturing Execution Systems”. In: *Proceedings of the IEEE* 104.5 (May 2016), pp. 1142–1154.

[57] J. Delsing. “Building Automation Systems from Internet of Things”. In: Presented as an 20th IEEE International Conference on Emerging Technologies and Factory Automation Keynote Presentation, Luxembourg, 2015.

[58] S. Cheshire and M. Krochmal. *RFC 6762: Multicast DNS*. Tech. rep. Feb. 2013.

[59] Arrowhead Consortium. *Arrowhead demo T4.2*. Tech. rep. 2014.

[60] F. Blomstedt and P. Olofsson. *Service Discovery DNS-SD-TSIG-SPDNS Version 1.3*. Tech. rep. The Arrowhead Consortium, May 2016.

[61] F. Blomstedt. *Service Discovery REST_WS-XML-SPSDTR Version 1.1*. Tech. rep. The Arrowhead Consortium, Apr. 2016.

[62] F. Blomstedt and P. Olofsson. *Service Discovery REST_WS-JSON-SPSDTR Version 1.1*. Tech. rep. The Arrowhead Consortium, Apr. 2016.

[63] F. Blomstedt. *ServiceRegistryBridge Version 1.1*. Tech. rep. The Arrowhead Consortium, Apr. 2016.

[64] T. Erl. *SOA: principles of service design*. Upper Saddle River, NJ: Prentice Hall, 2008.

[65] PLANTCockpit Consortium. *Technical Report - External Interface Specification, First Draft*. Tech. rep. Apr. 2012.

[66] J. Faist and M. Stetina. “Webservice-Ready Configurable Devices for Intelligent Manufacturing Systems”. In: *IFIP International Conference on Advances in Production Management Systems (APMS)*. Springer International Publishing, 2015.

[67] R. Ratovsky et al. *OpenAPI Specification Version 2.0*. Tech. rep. Open API Initiative, Sept. 2014.

[68] L. Dürkop et al. “Service-oriented architecture for the autoconfiguration of real-time Ethernet systems”. In: *3rd Annual Colloquium Communication in Automation (KommA)*. 2012.

[69] K. Ballinger et al. *Web Services Metadata Exchange 1.1 (WS-MetadataExchange)*. Tech. rep. Aug. 2008.

[70] W. Mahnke, S. Leitner, and M. Damm. *OPC Unified Architecture*. en. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[71] R. Henssen and M. Schleipen. “Interoperability between OPC UA and AutomationML”. en. In: *Procedia CIRP* 25 (2014), pp. 297–304. DOI: 10.1016/j.procir.2014.10.042.

[72] C. Jennings et al. *Media Types for Sensor Markup Language (SenML)*. Tech. rep. Apr. 2016.

[73] I. Building Performance Institute. *BPI-2100-S-2013 Standard for Home Performance-Related Data Transfer*. Tech. rep. June 2013.

[74] I. Building Performance Institute. *BPI-2200-S-2013 Standard for Home Performance-Related Data Collection*. Tech. rep. June 2013.

[75] Z. Shelby. *RFC 6690: Constrained RESTful Environments (CoRE) Link Format*. Tech. rep. Aug. 2012.

[76] Arrowhead Consortium. *WP 4 T4.1 Demonstrator (1st Generation): The Safe Home*. Tech. rep. 2014.

[77] L. Fumagalli et al. "Ontology-based modeling of manufacturing and logistics systems for a new MES architecture". In: *IFIP International Conference on Advances in Production Management Systems (APMS)*. Springer, 2014, pp. 192–200.

[78] E. Negri et al. "Ontology for Service-Based Control of Production Systems". In: *Advances in Production Management Systems: Innovative Production Management Towards Sustainable Growth*. Vol. 460. Cham: Springer International Publishing, 2015, pp. 484–492.

[79] eScop Consortium. *Semantic Workbench*. Tech. rep.

[80] J. Imtiaz et al. *D. 2.5 - Integrated Secure Plug&Work Framework*. Tech. rep. IoT@Work Consortium, July 2013.

[81] PLANTCockpit Consortium. *Technical Report - Data and Process Model, First Draft*. Tech. rep. Apr. 2012.

[82] PLANTCockpit Consortium. *Technical Report - Generic Alarms and Events Data Format*. Tech. rep. Mar. 2012.

[83] B. Hill. *Evaluation of efficient XML interchange (EXI) for large datasets and as an alternative to binary JSON encodings*. Tech. rep. DTIC Document, 2015.

[84] E. Negri et al. "Requirements and languages for the semantic representation of manufacturing systems". In: *Computers in Industry* 81 (Sept. 2016), pp. 55–66.

[85] D. Rotondi et al. *D1.1 - State of the Art and Functional Requirements in Manufacturing and Automation*. Tech. rep. IoT@Work Consortium, June 2010.

[86] PLANTCockpit Consortium. *Technical Report - Persistency and Synchronization Model*. Tech. rep. Apr. 2011.

[87] eScop Consortium. *Orchestration Layer Training Material*. Tech. rep.

[88] F. Galiegue, K. Zyp, and G. Court. *JSON Schema: core definitions and terminology*. Tech. rep. Internet Engineering Task Force, Jan. 2013.

[89] F. Galiegue, K. Zyp, and G. Court. *JSON Schema: interactive and non interactive validation*. Tech. rep. Internet Engineering Task Force, Jan. 2013.

[90] J. Eliasson. "Arrowhead Historian System". In: Presented at the Arrowhead Budapest Meeting, Mar. 2015.

[91] PLANTCockpit Consortium. *PLANTCockpit White Paper*. Tech. rep. Aug. 2012.

[92] J. Imtiaz and J. Jasperneite. “Scalability of OPC-UA down to the chip level enables “Internet of Things"”. In: 11th International Conference on Industrial Informatics (INDIN), July 2013, pp. 500–505.

[93] *OASIS Security Services (SAML) TC*. Tech. rep.

[94] S. Iarovyi et al. “Representation of manufacturing equipment and services for OKD-MES: from service descriptions to ontology”. In: *13th International Conference on Industrial Informatics (INDIN)*. 2015.

[95] O. Ben-Kiki, C. Evans, and B. Ingerson. *YAML Ain't Markup Language (YAML) (tm) Version 1.2*. Tech. rep. YAML.org, Sept. 2009.

[96] D. Box et al. *Simple Object Access Protocol (SOAP) 1.1*. W3C Note. World Wide Web Consortium, May 2000.

[97] R. Fielding and J. Reschke. *RFC 7235: Hypertext Transfer Protocol (HTTP/1.1): Authentication*. Tech. rep. June 2014.

[98] E. Rescorla. *RFC 2818: HTTP over TLS*. Tech. rep. May 2000.

[99] O. Severa et al. *eScopRTU with Service Manager*. Tech. rep. eScop Consortium.

[100] Z. Shelby, K. Hartke, and C. Bormann. *RFC 7252: The Constrained Application Protocol (CoAP)*. Tech. rep. June 2014.

[101] H. Derhamy. “Arrowhead Transparency - Protocol Translation”. In: Presented at the Arrowhead Budapest Meeting, Mar. 2015.

[102] Q. Mahmoud, ed. *Middleware for communications*. Chichester, England ; Hoboken, NJ: J. Wiley & Sons, 2004.

[103] M. Richards et al. *Java message service*. 2nd ed. Sebastopol, CA: O'Reilly, 2009.

[104] A. Skou et al. *Deliverable D5.3 of WP 5*. Tech. rep. Arrowhead Consortium, June 2014.

[105] OASIS. *Advanced Message Queuing Protocol (AMQP) Version 1.0*. Ed. by R. Godfrey, D. Ingham, and R. Schloming. 2012.

[106] J. Luzuriaga et al. “A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks”. In: *12th Annual Conference on Consumer Communications and Networking Conference (CCNC)*. 2015.

[107] M. Richards. “Understanding the Differences between AMQP & JMS”. In: *Proceedings of the No Fluff Just Stuff ™ Java Conference Series*. 2011.

[108] C. Le Pape et al. *Deliverable D1.3 of WP1*. Tech. rep. Arrowhead Consortium, June 2014.

[109] V. Gazis et al. “A survey of technologies for the internet of things”. In: *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2015.

[110] P. Saint-Andre. *RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core*. Tech. rep. Mar. 2011.

[111] R. Jeyaraman et al. *Understanding Devices Profile for Web Services, Web Services Discovery, and SOAP-over-UDP*. Tech. rep. Microsoft Corporation, Sept. 2008.

[112] R. Zurawski, ed. *The industrial communication technology handbook*. Industrial information technology series. Boca Raton: Taylor & Francis/CRC Press, 2005.

[113] J. Moreau et al. *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*. W3C Recommendation. World Wide Web Consortium, Apr. 2007.

[114] D. Hästbacka et al. "Device status information service architecture for condition monitoring using OPC UA". In: *19th International Conference on Emerging Technology and Factory Automation (ETFA)*. 2014.

[115] P. Bellavista and M. Ornato. *Arrowhead D3.3 Appendix 3.3b PO 3.3-002 and PO 3.3-004 of Task 3.3 Details about First Generation Pilot Results*. Tech. rep. June 2014.

[116] Arrowhead Consortium. *WP3.1.2 PO 002 - Requirements definition - Communication and services*. Tech. rep. Arrowhead Consortium, June 2014.

[117] S. Bocchio and M. Ornato. *Arrowhead D3.3 Appendix 3.1.1.c PO 002 of Task 3.1.1 Task and concepts*. Tech. rep. June 2014.

[118] D. Kleyko. *System-of-Systems Design (SoSDD) LTU Core Framework Description*. Tech. rep. Jan. 2016.

[119] A. Houyou et al. *D2.2- Bootstrapping Architecture*. Tech. rep. IoT@Work Consortium, June 2011.

[120] S. Gusmeroli, S. Piccione, and D. Rotondi. "A capability-based security approach to manage access control in the Internet of Things". In: *Mathematical and Computer Modelling* 58.5-6 (Sept. 2013), pp. 1189–1205.

[121] K. Fischer and J. Gesner. "Security architecture elements for IoT enabled automation networks". In: *IEEE 17th Conference on Emerging Technologies and Factory Automation (ETFA)*. Sept. 2012.

[122] PLANTCockpit Consortium. *Project Final Report*. Tech. rep. June 2014.

[123] F. Blomstedt and P. Olofsson. *Orchestration System Version 1.1*. Tech. rep. The Arrowhead Consortium, Apr. 2016.

[124] C. Chrysoulas and O. Jansson. *Arrowhead System Description (SysD) - Translation System Version 0.4*. Tech. rep. The Arrowhead Consortium, Sept. 2016.

[125] J. Krimm et al. *DELIVERABLE D8.2 of Work Package 8: Common Service Framework - Generation 1 Version 1.1*. Tech. rep. The Arrowhead Consortium, Jan. 2014.

[126] C. Lesjak et al. "ESTADO-Enabling smart services for industrial equipment through a secured, transparent and ad-hoc data transmission online". In: *IEEE 9th International Conference for Internet Technology and Secured Transactions (ICITST)*. 2014.

[127] G. Sadrollah et al. “A distributed framework for supporting 3D swarming applications”. In: *International Conference on Computer and Information Sciences (ICCOINS)*. 2014.

[128] J. Reschke. *RFC 7617: The 'Basic' HTTP Authentication Scheme*. Tech. rep. Sept. 2015.

[129] eScop Consortium. *D2.4 General specification and design of eScop reference architecture*. Tech. rep. Sept. 2014.

[130] P. Simone et al. *D6.1 Test strategy*. Tech. rep.

[131] *Part 12: Discovery*. R1.03. OPC Foundation. July 2015.

[132] *Part 4: Services*. R1.03. OPC Foundation. Oct. 2015.

[133] *Part 4: Services*. R1.03. OPC Foundation. July 2015.

[134] *Part 7: Profiles*. R1.03. OPC Foundation. Dec. 2015.

[135] A. Ismail and W. Kastner. “A middleware architecture for vertical integration”. In: *2016 1st International Workshop on Cyber-Physical Production Systems (CPPS)*. Apr. 2016.

[136] *XML Schema Part 2: Datatypes Second Edition*. W3C Note. Oct. 2004.

[137] *Part 6: Services*. R1.03. OPC Foundation. July 2015.

[138] *Part 2: Security Model*. R1.03. OPC Foundation. Nov. 2015.

[139] J. Zerbst et al. “Towards an Adapted Classification Methodology for Graded Security Approaches in EPU Architectures”. In: *Proceedings of the International Council on Large Electric Systems (CIGRE)*. Apr. 2013.

[140] Z. Ou. “Structured peer-to-peer networks: Hierarchical architecture and performance evaluation.” In: *Dissertation, Acta Universitatis Ouluensis C Technica 361, Department of Electrical and Information Engineering, University of Oulu, Finland.* ().

[141] S. Androutsellis-Theotokis and D. Spinellis. “A survey of peer-to-peer content distribution technologies”. In: *ACM Computing Surveys (CSUR)* 36.4 (2004), pp. 335–371.

[142] V. Altmann et al. “A DHT-Based Scalable Approach for Device and Service Discovery”. In: *12th IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*. Aug. 2014, pp. 97–103.

[143] E. K. Lua et al. “A survey and comparison of peer-to-peer overlay network schemes”. In: *Communications Surveys & Tutorials, IEEE* 7.2 (2005), pp. 72–93.

[144] P. Wang, B. Aslam, and C. C. Zou. “Peer-to-Peer botnets: the next generation of botnet attacks”. In: *Electrical Engineering* (2010), pp. 1–25.

[145] B. Pourebrahimi, K. Bertels, and S. Vassiliadis. “A survey of peer-to-peer networks”. In: *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc*. Vol. 2005. Citeseer, 2005.

[146] C. Xie et al. “Analysis of hybrid P2P overlay network topology”. en. In: *Computer Communications* 31.2 (Feb. 2008), pp. 190–200. DOI: 10.1016/j.comcom.2007.08.014.

[147] P. Maymounkov and D. Mazieres. “Kademlia: A peer-to-peer information system based on the xor metric”. In: *Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[148] Z. Xu, X. He, and L. Bhuyan. “Efficient file sharing strategy in DHT based P2P systems”. In: *24th IEEE International Performance, Computing, and Communications Conference (IPCCC)*. Apr. 2005, pp. 151–158. DOI: 10.1109/PCCC.2005.1460541.

[149] C. Lesniewski-laas. “A Sybil-Proof One-Hop DHT”. In: *Workshop on Social NetworkSystems,Glasgow,Scotland*. 2008.

[150] M. S. Artigas et al. “Cyclone: A novel design schema for hierarchical dhts”. In: *5th IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*. IEEE, 2005, pp. 49–56.

[151] H. G. Ngo. “From inter-connecting P2P overlays to co-operating P2P systems”. PhD thesis. Université Nice Sophia Antipolis; Hanoi University of sciences (Hanoi), 2013. URL: https://tel.archives-ouvertes.fr/tel-00937695/.

[152] D. Malkhi, M. Naor, and D. Ratajczak. “Viceroy: A scalable and dynamic emulation of the butterfly”. In: *Proceedings of the 21st annual symposium on Principles of distributed computing*. ACM, 2002, pp. 183–192.

[153] A. Datta and K. Aberer. “The challenges of merging two similar structured overlays: A tale of two networks”. In: *Lecture notes in computer science* 4124 (2006), p. 7.

[154] S. Ratnasamy et al. “A Scalable Content-addressable Network”. In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM ’01. San Diego, California, USA: ACM, 2001, pp. 161–172. DOI: 10.1145/383059.383072.

[155] N. S. Evans, C. Gauthierdickey, and C. Grothoff. “Routing in the dark: Pitch black”. In: *In Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC). IEEE Computer Society*. 2007.

[156] I. Stoica et al. “Chord: a scalable peer-to-peer lookup protocol for Internet applications”. In: *IEEE/ACM Transactions on Networking* 11.1 (Feb. 2003), pp. 17–32. DOI: 10.1109/TNET.2002.808407.

[157] B. Zhao et al. “Tapestry: A Resilient Global-Scale Overlay for Service Deployment”. en. In: *IEEE Journal on Selected Areas in Communications* 22.1 (Jan. 2004), pp. 41–53. DOI: 10.1109/JSAC.2003.818784.

[158] D. Karger et al. “Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web”. In: *Proceedings of the 29th annual ACM symposium on Theory of computing*. ACM, 1997, pp. 654–663.

[159] C. G. Plaxton, R. Rajaraman, and A. W. Richa. “Accessing Nearby Copies of Replicated Objects in a Distributed Environment”. In: *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*. SPAA ’97. Newport, Rhode Island, USA: ACM, 1997, pp. 311–320. DOI: 10.1145/258492.258523.

[160] A. Rowstron and P. Druschel. “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems”. In: *Middleware 2001*. Springer, 2001, pp. 329–350.

[161] E. Gamma, ed. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley professional computing series. Reading, Mass: Addison-Wesley, 1995. ISBN: 0201633612.

[162] K. Aberer et al. “P-Grid: a self-organizing structured P2P system”. In: *ACM SIGMOD Record* 32.3 (2003), pp. 29–33.

[163] R. Schmidt. “The P-Grid SystemOverview”. In: *School of Computer and Communication Sciences, Ecole Polytechnique Federale de Lausanne (EPFL) CH-1015 Lausanne, Switzerland, Tech. Rep* (2007).

[164] I. Clarke et al. “Freenet: A distributed anonymous information storage and retrieval system”. In: *Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 46–66.

[165] B. Cohen and A. Norberg. “The BitTorrent Protocol Specification”. Oct. 2013. URL: http://www.bittorrent.org/beps/bep_0003.html.

[166] A. Bhakuni, P. Sharma, and R. Kaushal. “Free-rider detection and punishment in BitTorrent based P2P networks”. In: *Proceedings of the 2014 IEEE International Advance Computing Conference (IACC)*. IEEE, 2014, pp. 155–159.

[167] D. Erman, K. De Vogeleer, and A. Popescu. “On piece selection for streaming bittorrent”. In: *Fifth Swedish National Computer Networking Worshop (SNCNW2008)* 16 (2008).

[168] K. Birman. “CS5412: Torrents and Tit-For-Tat”. 2012. URL: http://www.cs.cornell.edu/Courses/cs5412/2012sp/slides/VI%20-%20Torrents%20and%20Tit%20for%20Tat.pdf.

[169] S. Ghosh. “The BitTorrent Protocol”. URL: https://www.cs.uiowa.edu/~ghosh/bittorrent.ppt (visited on 07/09/2015).

[170] B. Y. Zhao et al. “Brocade: Landmark routing on overlay networks”. In: *Peer-to-Peer Systems*. Springer, 2002, pp. 34–44.

[171] L. Garces-Erice et al. “Hierarchical peer-to-peer systems”. In: *Parallel Processing Letters* 13.04 (2003), pp. 643–657.

[172] Z. Xu, R. Min, and Y. Hu. "HIERAS: a DHT based hierarchical P2P routing algorithm". In: *Parallel Processing, 2003. Proceedings. 2003 International Conference on*. IEEE, 2003, pp. 187–194.

[173] B. Yang and H. Garcia-Molina. "Designing a super-peer network". In: *Proceedings of the 19th International Conference on Data Engineering*. IEEE, 2003, pp. 49–60.

[174] S. Zoels, Z. Despotovic, and W. Kellerer. "Cost-based analysis of hierarchical dht design". In: *6th IEEE International Conference on Peer-to-Peer Computing (P2P 2006)*. IEEE, 2006, pp. 233–239.

[175] S. Ratnasamy et al. "Topologically-aware overlay construction and server selection". In: *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 3. 2002, 1190–1199 vol.3.

[176] S. Zoels, Z. Despotovic, and W. Kellerer. "On hierarchical DHT systems An analytical approach for optimal designs". en. In: *Computer Communications* 31.3 (Feb. 2008), pp. 576–590. DOI: 10.1016/j.comcom.2007.08.033.

[177] L. Cheng. "Bridging distributed hash tables in wireless ad-hoc networks". In: *IEEE Global Telecommunications Conference (GLOBECOM'07)*. IEEE, 2007, pp. 5159–5163.

[178] L. Liquori, C. Tedeschi, and F. Bongiovanni. "Babelchord: a social tower of DHT-based overlay networks". In: *IEEE Symposium on Computers and Communications (ISCC 2009)*. July 2009, pp. 307–312. DOI: 10.1109/ISCC.2009.5202345.

[179] L. Liquori et al. *Synapse: A Scalable Protocol for Interconnecting Heterogeneous Overlay Networks*. Tech. rep. 2009.

[180] V. Ciancaglini et al. "An extension and cooperation mechanism for heterogeneous overlay networks". In: *NETWORKING 2012 Workshops*. Springer, 2012, pp. 10–18.

[181] V. Ciancaglini, L. Liquori, and G. N. Hoang. "Towards a Common Architecture to Interconnect Heterogeneous Overlay Networks". In: *17th IEEE International Conference on Parallel and Distributed Systems*. IEEE, Dec. 2011, pp. 817–822. DOI: 10.1109/ICPADS.2011.139.

[182] G. N. Hoang et al. "A backward-compatible protocol for inter-routing over heterogeneous overlay networks". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 649–651.

[183] H. G. Ngo, L. Liquori, and C. H. Nguyen. "Backward-Compatible Cooperation of Heterogeneous P2P Systems". In: *Distributed Computing and Networking*. Springer, 2014, pp. 287–301.

[184] T. M. Shafaat, A. Ghodsi, and S. Haridi. "Dealing with network partitions in structured overlay networks". en. In: *Peer-to-Peer Networking and Applications* 2.4 (Dec. 2009), pp. 334–347. DOI: 10.1007/s12083-009-0037-7.

[185] M. Kwon and S. Fahmy. “Synergy: an overlay internetworking architecture”. In: *Proceedings of the 14th IEEE International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2005, pp. 401–406.

[186] X. Liao et al. “AnySee: Peer-to-Peer Live Streaming”. In: *Proceedings of the 25TH IEEE International Conference on Computer Communications (INFOCOM)*. Apr. 2006, pp. 1–10. DOI: 10.1109/INFOCOM.2006.288.

[187] V. Ciancaglini. “From key-based to content-based routing: system interconnection and video streaming applications”. PhD thesis. Université de Nice - Sophia Antipolis, Oct. 2013.

[188] J. Konishi, N. Wakamiya, and M. Murata. “Proposal and Evaluation of a Cooperative Mechanism for Pure P2P File Sharing Networks”. In: *Proceedings of the 2nd International Workshop on Biologically Inspired Approaches to Advanced Information Technology (BioADIT)*. Springer Berlin Heidelberg, Jan. 2006, pp. 33–47.

[189] I. Stoica et al. “Internet indirection infrastructure”. In: *IEEE/ACM Transactions on Networking* 12.2 (Apr. 2004), pp. 205–218.

[190] F. Johnsen and T. Hafsøe. “Adapting WS-Discovery for use in tactical networks”. In: *16th International Command and Control Research and Technology Symposium*. June 2011.

[191] R. Kyusakov, J. Eliasson, and J. Delsing. “Efficient structured data processing for web service enabled shop floor devices”. In: *2011 IEEE International Symposium on Industrial Electronics*. June 2011, pp. 1716–1721.

[192] T. Kothmayr et al. “Machine ballets don’t need conductors: Towards scheduling-based service choreographies in a real-time SOA for industrial automation”. In: *Proceedings of the 19th IEEE Emerging Technology and Factory Automation (ETFA)*. Sept. 2014, pp. 1–6.

[193] P. Hunt et al. “ZooKeeper: Wait-free Coordination for Internet-scale Systems.” In: *USENIX Annual Technical Conference*. Vol. 8. 2010.

[194] G. Hohpe and B. Woolf. *Enterprise integration patterns: designing, building, and deploying messaging solutions*. The Addison-Wesley signature series. Boston: Addison-Wesley, 2004. ISBN: 978-0-321-20068-6.

[195] K. Stouffer et al. *NIST Special Publication 800-82 Revision 2: Guide to Industrial Control Systems (ICS) Security*. Tech. rep. National Institute of Standards and Technology, May 2015.

[196] E. Jones. *Preventing server overload: limit requests being processed*. July 22, 2017. URL: http://www.evanjones.ca/prevent-server-overload.html (visited on 08/22/2017).

[197] H. Deng and F. Junqueira. *ZooKeeper SSL User Guide*. Atlassian Confluence. July 24, 2015. URL: https://cwiki.apache.org/confluence/display/ZOOKEEPER/ZooKeeper+SSL+User+Guide (visited on 04/06/2017).

[198] R. Cupek et al. “Performance evaluation of redundant OPC UA architecture for process control”. In: *Transactions of the Institute of Measurement and Control* (Sept. 2015).

[199] S. Mackrory. *How-to: Use Apache ZooKeeper to Build Distributed Apps (and Why)*. Cloudera Inc. Feb. 14, 2013. URL: http://blog.cloudera.com/blog/2013/02/how-to-use-apache-zookeeper-to-build-distributed-apps-and-why/ (visited on 04/04/2017).

[200] *HDFS High Availability Using the Quorum Journal Manager*. URL: https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html (visited on 11/17/2016).

[201] F. Palm et al. “Open source as enabler for OPC UA in industrial automation”. In: *20th IEEE International Conference on Emerging Technology and Factory Automation (ETFA)*. IEEE, Sept. 2015, pp. 1–6. DOI: 10.1109/ETFA.2015.7301562.

[202] D. Grossmann et al. “OPC UA server aggregation - The foundation for an internet of portals”. In: *19th IEEE International Conference on Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–6. DOI: 10.1109/ETFA.2014.7005354.

[203] I. Seilonen et al. “Aggregating OPC UA servers for monitoring manufacturing systems and mobile work machines”. In: *21st IEEE International Conference on Emerging Technology and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–4. DOI: 10.1109/ETFA.2016.7733739.

[204] J. Asikainen. “OPC UA Java History Gateway with Inherent Database Integration”. MA thesis. Aalto University, 2014.

[205] D. Singh and C. K. Reddy. “A survey on platforms for big data analytics”. In: *Journal of Big Data* 2.1 (2015), p. 8.

[206] S. Gilbert and N. Lynch. “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services”. In: *Acm Sigact News* 33.2 (2002), pp. 51–59.

[207] M. J. Fischer, N. A. Lynch, and M. S. Paterson. “Impossibility of distributed consensus with one faulty process”. en. In: *Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*. ACM Press, 1983. DOI: 10.1145/588058.588060.

[208] A. Ismail and W. Kastner. “Coordinating Redundant OPC UA Servers”. In: *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sept. 2017, pp. 1–8.

[209] *Apache Curator Recipes: Distributed Queue*. Version 4.0.0. Apache Software Foundation. July 23, 2017. URL: https://curator.apache.org/curator-recipes/distributed-queue.html (visited on 08/21/2017).

[210] *Apache Curator: Tech Note 4*. Aug. 8, 2013. URL: https://cwiki.apache.org/confluence/display/CURATOR/TN4.

[211] S. Chintapalli et al. "PaceMaker: When ZooKeeper Arteries Get Clogged in Storm Clusters". In: *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, June 2016, pp. 448–455. DOI: 10.1109/CLOUD.2016.0066.

[212] P. Adolphs et al. *Reference Architecture Model Industrie 4.0 (RAMI4.0)*. VDI/VDE Society Measurement and Automatic Control (GMA). July 2015.

[213] B. Rexroth and M. Hankel. *Reference Architecture Model Industrie 4.0 (RAMI4.0)*. VDI/VDE Society Measurement and Automatic Control (GMA). Version 1.0. Apr. 2015.

[214] J. Skodzik et al. "Hartkad: A hard real-time kademlia approach". In: *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*. IEEE, 2014, pp. 309–314.

[215] B. Konieczek et al. "HaRTKad: A P2P-based concept for deterministic communication and its limitations". In: *Proceedings of the 2016 IEEE Symposium on Computers and Communication (ISCC)*. IEEE, 2016, pp. 1157–1162.