FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Dissertation

Dynamically Scalable Fog Architectures

Dominic Henze

FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Forschungs- und Lehreinheit 1
Angewandte Softwaretechnik

Dynamically Scalable Fog Architectures

Dominic Henze

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Jens Großklags, Ph.D.
Prüfende/-r der Dissertation:	1. Prof. Dr. Bernd Brügge
	2. Prof. Dr. Horst Lichter

Die Dissertation wurde am 01.07.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 12.09.2020 angenommen.

Abstract

Recent advances in mobile connectivity as well as increased computational power and storage in sensor devices have given rise to a new family of software architectures with challenges for data and communication paths as well as architectural reconfigurability at runtime. Established in 2012, Fog Computing describes one of these software architectures, but lacks a common accepted definition. This manifests itself among others in the missing support for mobile applications as well as dynamically changing runtime configurations.

This dissertation provides a framework that formalizes Fog Computing and adds support for dynamic and scalable Fog Architectures. The framework is called xFog (E**x**tension for **Fog** Computing) and models Fog Architectures based on mathematical sets and graphs. xFogPlus, one part of xFog, enables dynamic and scalable Fog Architectures to dynamically add new components or layers. Additionally, xFogPlus provides a View concept which allows stakeholders to focus on different levels of abstraction. These formalizations establish the foundation for new concepts in the area of Fog Computing. One such concept, xFogStar, provides a workflow to find the best service configuration based on quality of service parameters.

xFog has been applied in eight case studies covering different application domains ranging from smart environments, health, and metrology to gaming. They investigate the applicability of dynamic Fog Components, scalable Fog Architectures, and the service provider selection at runtime.

The case studies successfully demonstrated the feasibility of the formalization provided by xFog, the dynamic change of Fog Architectures by adding new components and layers at runtime, and the applicability of the workflow to establish the best service configuration.

Zusammenfassung

Die jüngsten Fortschritte in mobiler Konnektivität und Sensorgeräten mit erhöhter Rechenleistung und Speicherkapazität führten zu einer neuen Familie von Softwarearchitekturen mit Herausforderungen an Daten- und Kommunikationspfade und an die architektonische Rekonfigurierbarkeit zur Laufzeit. 2012 wurde unter der Bezeichnung Fog Computing eine derartige Softwarearchitektur beschrieben, die aber über keine allgemein anerkannte Definition verfügt. Dies äußert sich unter anderem in der fehlenden Unterstützung für mobile Anwendungen sowie dynamische Änderungen der Laufzeitkonfigurationen.

Diese Dissertation stellt ein Framework bereit, das Fog Computing formalisiert und eine Basis für dynamische und skalierbare Fog Architekturen definiert. Das Framework namens xFog (Extension for **Fog** Computing) modelliert Fog Architekturen aufbauend auf mathematischen Mengen und Graphen. xFogPlus, ein Teil von xFog, beschreibt dynamische und skalierbare Fog Architekturen, um neue Komponenten oder Layer dynamisch hinzuzufügen. Zusätzlich liefert xFogPlus ein View-Konzept mit dem sich Stakeholder auf verschiedene Abstraktionsebenen fokussieren können. Diese Formalisierungen bilden die Grundlage für neue Konzepte im Bereich Fog Computing. Eines dieser Konzepte, xFogStar, definiert einen Workflow, um die beste Servicekonfiguration basierend auf Quality of Service Parametern zu finden.

xFog wurde in acht Fallstudien aus verschiedenen Anwendungsbereichen angewendet, die von intelligenten Umgebungen über Gesundheitswesen und Messtechnik bis hin zu Spielen reichen. Sie untersuchen die Anwendbarkeit dynamischer Fog Komponenten, skalierbarer Fog Architekturen und die Auswahl von Dienstanbietern zur Laufzeit.

Die Fallstudien zeigten erfolgreich die Umsetzbarkeit der von xFog bereitgestellten Formalisierungen, die dynamische Änderung von Fog Architekturen durch Hinzufügen neuer Komponenten und Layer während der Laufzeit und die Anwendbarkeit des Workflows, um die beste Servicekonfiguration zu ermitteln.

Acknowledgements

First and foremost, I would like to thank Professor Bernd Brügge for supporting me all the way from the moment we first met in the Dolli6 practical course during my studies. He did not only nurture my passion for modeling and software engineering, but also offered me the opportunity to join the Chair for Applied Software Engineering. I am thankful for his passion for teaching which convinced me to further pursue my scientific carrier–as a researcher but also an educator, the productive discussions, and his trust in my management skills. I also want to express my gratitude to Professor Horst Lichter who stepped in as my second advisor, gave me the opportunity to present my research, and provided valuable input already in the first meeting. I would like to thank Professor Vivian Loftness and Professor Volker Hartkopf for giving me the unique opportunity to join the Intelligent Workplace and collaborate on research projects. I thank the German Ministry of Education and Research for supporting this dissertation with the Software Campus initiative.

I would like to thank the entire LS1 family: You make the chair the place I know and love. I thank Monika Markel, Helma Schneider, and Uta Weber for keeping the chair up and running; our system administrators, particularly Matthias Linhuber and Florian Angermeier for realizing all our crazy ideas; Jan Ole Johanßen for his scientific guidance, but also the endless amount of shared coffee breaks; Andreas Seitz for our discussions and research cooperations; Nadine von Frankenberg und Ludwigsdorff for her endless positivity since our time as undergraduates; Mariana Avezum for the mutual encouragement while writing our dissertations, and the team led by Professor Stephan Jonas, who are an amazing addition to our LS1 family.

I would like to thank the iPraktikum and the multimedia team, in particular Lukas Alperowitz and Paul Schmiedmayer for their motivation and the ideas which made the course what it is today. Last but most definitely not least, I would like to thank Dora Dzvonyar, who did not only share an office with me, but also her passion for education and countless laughs.

I thank my friends for their support, but also the distraction from work, which was particularly needed during the months of writing.

Finally, I am deeply grateful for the endless support and encouragement of my family–without you, this dissertation would have never happened.

To my family and my friends who supported me all the way.

Contents

Conventions

This dissertation uses American English, except for direct quotes. Direct quotes are highlighted in ”double quotes” and changes to them are indicated by [squared brackets]. Text in *italic* font is used whenever referring to labels, names of figures, tables, new concepts, or when addressing provided definitions. We use the `Typewriter` font to refer to parts of UML diagrams such as classes, objects, and activities. Concepts that are introduced or redefined within this dissertation are written in uppercasing throughout the entire document.

Mathematical definitions, equations, corollaries, or proofs are highlighted using boxes of the following styles:

Definition 1: Title

A new definition.

Equation 1: Title

A new equation.

Corollary 1: Title

A new corollary.

Proof 1: Title

A proof.

In the mathematical context, variables in lower case are considered instances, while upper cases are used for concepts.

Additionally, to highlight important aspects such as research goals, we use the following boxes:

> **Research Goal:** A research goal.

This document contains colored figures and should therefore be printed in color. All figures are checked to be readable in greyscale, but might lose clarity.

Whenever trademarked names, such as company names, are used, they are used for identification only and are followed by links in footnotes forwarding to the official homepage.

Chapter 1

Introduction

"Architects need to anticipate changes, changes in the environment in which the system under development will be deployed, which will in turn trigger requests for change or evolution."

— PHILIPPE KRUCHTEN [81]

With the constant increase of computational power and available storage, mobile devices get more and more involved in distributed systems, which are "collections of independent computers that appear to be one single system to users" [139]. Nevertheless, mobile devices will always be resource poor in comparison to static hardware, as static hardware is not capped by properties such as heat dissipation or battery life [46, 119]. Therefore, mobile devices will always struggle with the most advanced media and data analysis.

Mobile cloud computing was introduced to bridge this gap and combines mobile computing with cloud computing to leverage the computational power of the cloud for mobile devices [43, 113]. However, clouds are usually distant from the mobile devices and using them creates high latencies, which are insufficient for realtime applications such as augmented reality.

To address this issue, concepts such as Cloudlets, Edge Computing, and Fog Computing emerged. Satyanarayanan et al. described these concepts to utilize resource-rich components near the mobile device to offload computational intense tasks while having "low latency, one-hop, high-bandwidth wireless access" [119]. While Cloudlets use trusted, nearby components with excessive computational power, Edge Computing focuses on the entirety of the network trying to push services as close to the edge as possible [44, 118].

Bonomi et al. introduced Fog Computing as a three-layered software architecture containing a Cloud, Fog, and Edge layer [18]. These layers interact using subscriber models with one layer acting as the provider and the other one as its user. Accordingly, application scenarios such as dynamic vehicles, smart grids, distributed sensor networks, and smart environments can benefit from using Fog Computing.

This loose definition has led to many interpretations of Fog Computing as well as attempts to sharpen the definition. Nevertheless, there is no commonly accepted definition of what Fog Computing or a Fog Node is, and the difference to similar concepts such as Edge Computing is not defined [94].

Section 1.1 describes the motivation why Fog Computing requires further research in the context of software architectures. Section 1.2 summarizes the objectives this dissertation addresses and its relations to the developed framework. The dissertation's methodology is presented in Section 1.3 providing an overview of the research process, which is applied and described in Section 1.4. Finally, Section 1.5 gives an overview of the dissertation's structure.

1.1 Problem

Figure 1.1 and Figure 1.2 show two examples for Fog Architectures in the smart environment and smart city domains. They are strictly hierarchical with on singular cloud and a variety of edge devices on the outermost layers. In between those layers, several other layers compose the Fog.

In the smart environment example, the Fog consists of bigger and bigger groupings of edge devices directly related to those found in home environments: First, all edge devices are gathered in according rooms, the rooms in living units, and the living units in floors. Each house contains floors and is in a group with other houses depending on the property they are on. Finally, the cloud is represented by the holding. Fog Nodes can be placed along these groupings to promote different services.

In contrast, the smart city example contains dynamic vehicles as edge devices that are grouped based on the nearest Fog Node to their current location. These Fog Nodes can for example be placed at each crossing. Several crossings make up a street, the street belongs to a city, the city to a district, and finally the district to a state. These layers, represented by at least one Fog Node can offer different services based on their locality and availability of data.

Based on existing definitions, e. g. the original definition by Bonomi et al. [18] or the refined definition by Yi et al. [155], it remains unclear if the layers between the cloud and the edge are considered part of one overarching Fog layer or if the Fog contains an internal structure itself. Additionally, no information is provided on how new layers can be added to Fog Architectures. For example, in the smart city Fog Architecture, big cities might group the streets into city districts before gathering the data themselves.

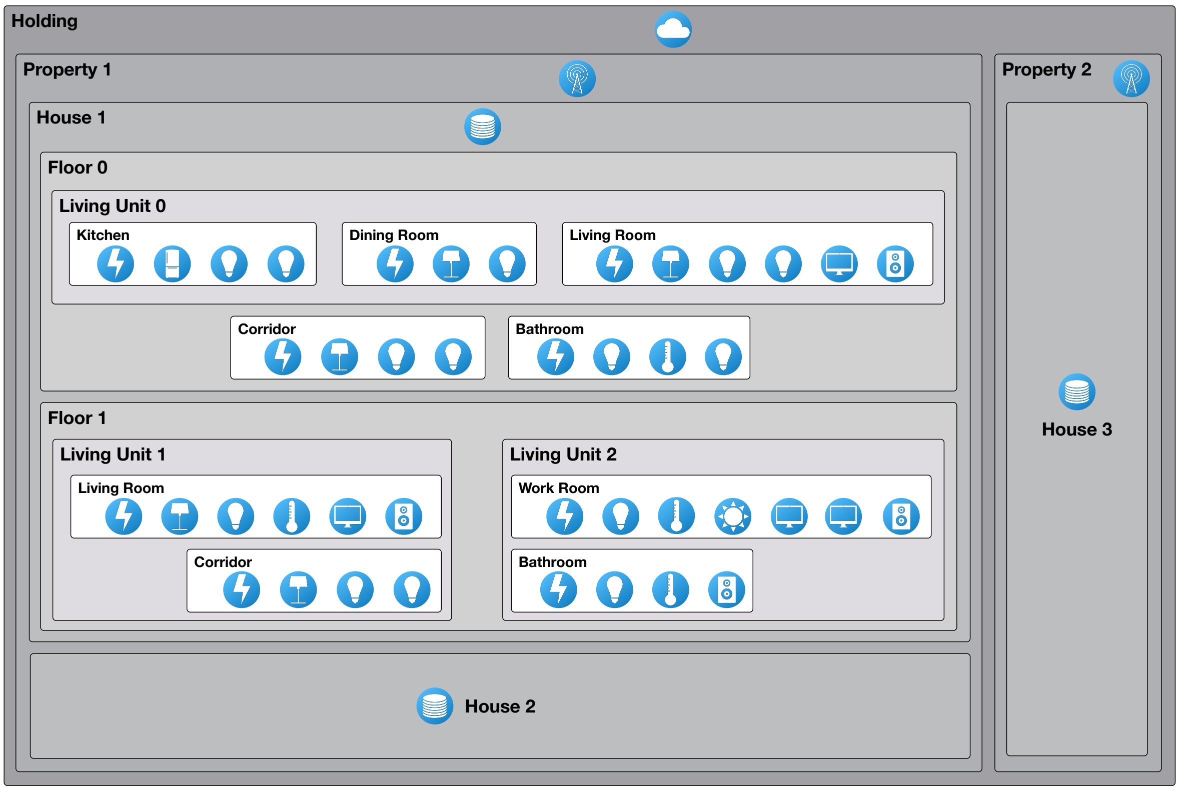


Figure 1.1: An example setup for a smart environment. All *Devices* are grouped into *Rooms*, each *Room* is in a *Living Unit*, which a *Floor* is composed of. A *House* contains *Floors* and stands on a *Property*, which can in turn contain other *Houses*. Finally, all *Properties* are in *Holdings*. This layering can also directly be mapped to a Fog Architecture with each container being an individual layer that aggregates its content.

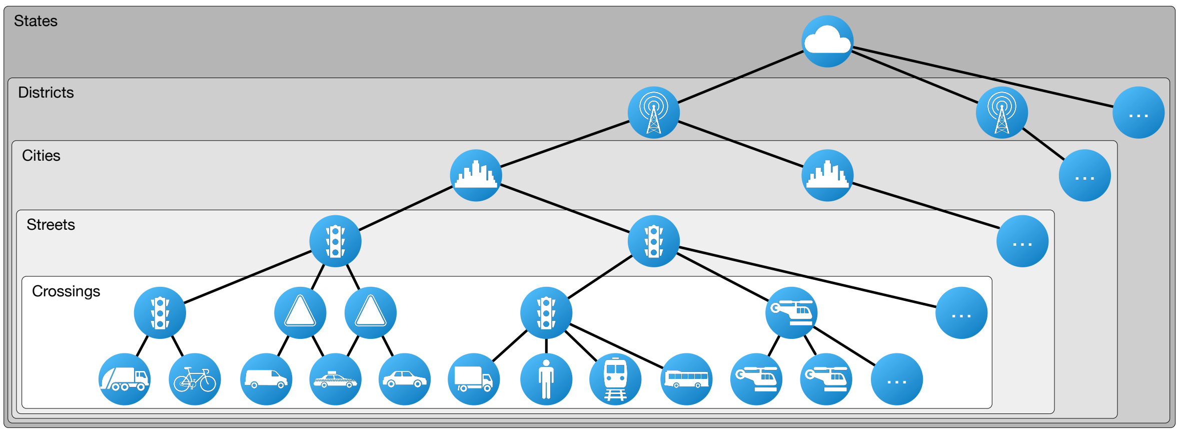


Figure 1.2: Infrastructure example with *Vehicles* deployed in the *Edge*. Each *Vehicle* registers at each *Crossing*, which are part of a *Street*. These *Streets* are within *Cities* that are in *Districts* that are in *States*.

Figure 1.3: A set of *Fog Components* which relate to a Fog Architecture, but missing their relations to each other as well as their layering.

Figure 1.3 poses another challenge. It shows a collection of internet of things devices, but lacks information on the devices' connections between each other. Based on Shaw et al. and Bass et al. the basic building blocks of software architectures are their components and connectors [13, 134]. Accordingly, it is crucial to know which of the presented devices are part of one Fog Architecture and which are not, drawing the boundaries of the software architecture. Additionally, Figure 1.3 does not include information about the different layers. To group the components which are part of the Fog Architecture to their according layer, we need to know the defining trades of each layer.

According to Yousefpour et al., the amount of devices and the dynamics increases closer to the edge of the network [158]. This is also supported by Bonomi et al. who identified the support for mobility as one of Fog Computing's core concepts [17, 18]. However, it is never mentioned how Fog Architectures handle dynamics, and thus components that join and leave a Fog Architecture at runtime.

Consider Figure 1.4, it shows an example with two Fog Nodes which are represented by the WIFI icons. Each node has a certain reach (dashed circle), within which other components can establish a connection to them. The other components are drones, labeled 1 to 11. At timestamp T1, which is indicated by the blue circles of the drones, some drones are within the Fog Nodes' reach and others are outside. The arrows indicate the directions and distances the drones travel until timestamp T2, which are shown as the greyish-blue circles. Thus, some drones that are currently inside the Fog Nodes' reach leave and others join.

This kind of continues, dynamic change is challenging for software architectures which are defined by their contained components and connectors.

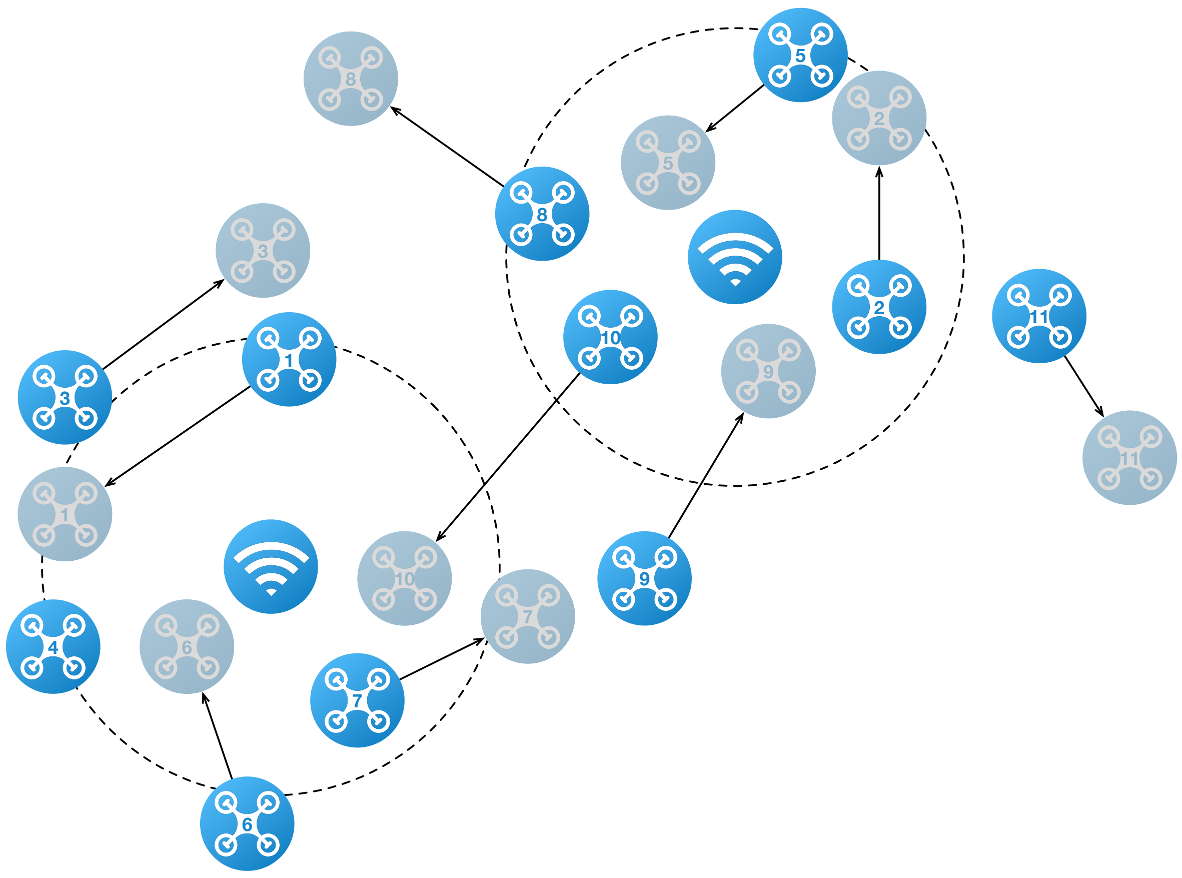


Figure 1.4: Dynamic Fog Architecture example with two Fog Nodes represented by WIFI icons. The Fog Nodes' reach, within which components can connect to them, is indicated by the dashed circles around the icons. The drones 1 to 11 are dynamic and can move in and out of the Fog Nodes' reach. The arrows from each drone indicate their target destination where they will arrive at timestamp T_2.

It blurs the boundaries of a Fog Architecture and requires constant updates of its description. Additionally, dynamics introduces complexity which can get challenging for stakeholders.

1.2 Research Objectives

As explained in the problem description, Fog Computing's lack of a common definition which leads to different researchers understanding and describing Fog Computing in different ways. To counteract these misunderstandings, this dissertation focuses on the creation of a formalized definition for Fog Computing based on software architectures and set theory, introducing the following goals:

> **Knowledge Goal 1:** Establish Fog Computing as a subclass of software architectures.

> **Technical Research Goal 1:** Define a framework that provides a formalized definition of Fog Computing.

These goals should be achieved with xFog, an e**x**tension for **Fog** Computing, and xFogCore which defines the foundations of the framework. These foundations are used to address two of the described challenges: Providing support for dynamic components and addressing the ambiguity of layers, in particular of the Fog.

> **Technical Research Goal 2:** Define an extension to the foundations of Fog Computing that supports components joining and leaving an architecture (*Dynamics*).

> **Technical Research Goal 3:** Define an extension to the foundations of Fog Computing that supports the process of adding and removing layers (*Scalability*).

These goals should be addressed with xFogPlus, one part of xFog that relies on the foundations introduced by xFogCore and formalizes dynamics and scalability in Fog Architectures.

The combination of both, xFogCore and xFogPlus, enables xFog to support a variety of advanced concepts, such as xFogStar. xFogStar is an extension that investigates the selection of service providers which is addressed with the following goal:

Technical Research Goal 4: Enable service provider selection in dynamic scalable Fog Architectures.

Finally, we want to investigate if and how xFog, its parts, and extension describe Fog Computing:

Knowledge Goal 2: Investigate the feasibility of xFog.

Knowledge Goal 3: Investigate the feasibility of xFogStar.

Therefore, we validate three different aspects: *Dynamic Fog Components*, *Scalable Fog Architectures*, and the *Service Provider Selection*. Each aspect is addressed by a multiple case study with cases from different domains.

1.3 Methodologies

In this section, we introduce the methodologies this dissertation uses. We present the chosen research approach *Design Science* that we base the structure of this dissertation on. Second, we present the validation strategy *Multiple Case Study*, which we use to validate the three treatment designs introduced in Chapter 3, Chapter 4, and Chapter 5. The validation is shown in Chapter 6.

1.3.1 Design Science

We base our description of *Design Science* on Hevner et al. [67] and several publications by Wieringa [147, 148, 149, 150]. According to Wieringa and Hevner et al., *Design Science* is the design and investigation of artifacts in a context to interact with a problem. Cross refers to design science as "[...] an explicitly organised, rational and wholly systematic approach to design; not just the utilisation of scientific knowledge of artefacts, but design in some sense a scientific activity itself" [36].

The involved artifacts have to be seen in a broad sense, ranging from hardware and software components to services, techniques or even structures, trying to improve a problem in its given problem context. Therefore, the context can be as broad as the artifacts themselves, i. e., other software components, stakeholders which affect or are affected by the artifact, goals that should be achieved, norms that need to be fulfilled, or even available budget.

Design Science has two categories of goals: knowledge goals and technical research goals. While knowledge goals are addressed by knowledge questions, technical research goals are refined to design problems. Knowledge questions try to find answers that universally hold true with one single answer, and therefore call for a change of our knowledge [150, 149]. Design problems, on the other hand, call for a change of artifacts, for instance a new way to store electrical energy or a more efficient database. Thus, design problems can be addressed by different artifacts in different ways, heavily relying on the given context. Both activities are inherently connected. New answers to knowledge questions can lead to new design problems and the result and evaluation of design problems can bring up new knowledge questions and answers. Therefore, the *Design Science* process is iteratively striving for improvements of the status-quo.

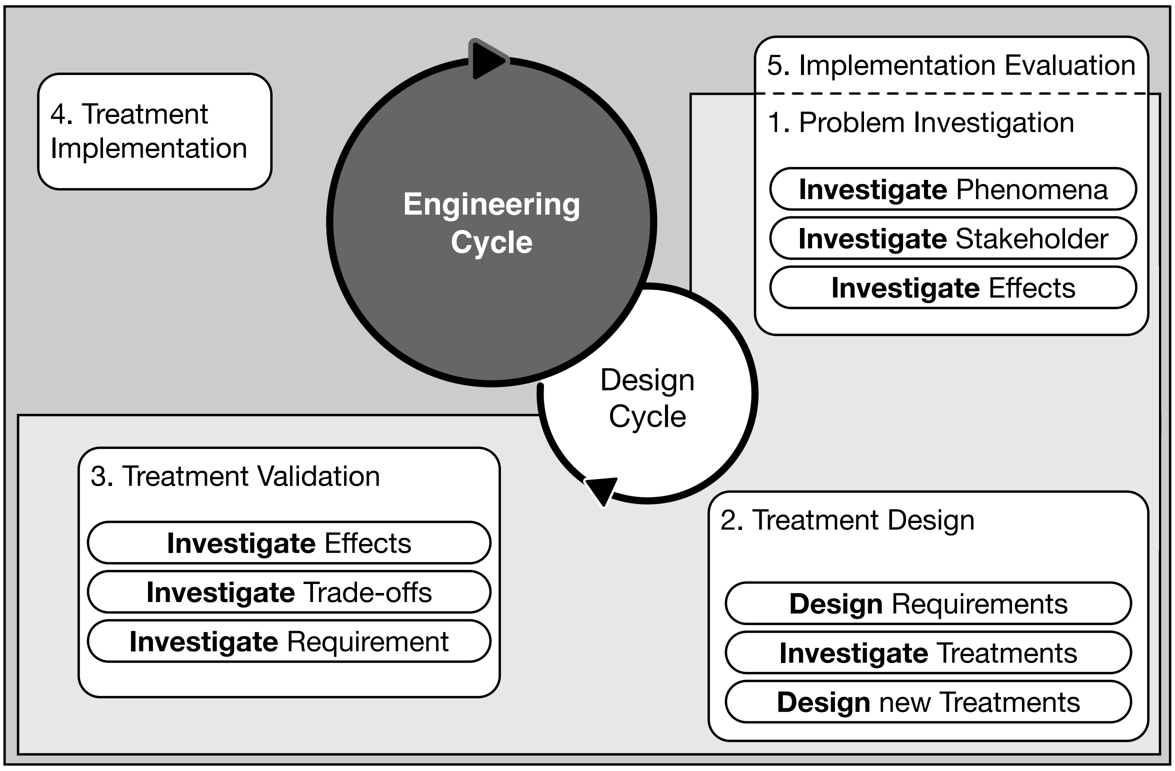

4. Treatment Implementation
Engineering Cycle
5. Implementation Evaluation
1. Problem Investigation
Investigate Phenomena
Investigate Stakeholder
Investigate Effects
Design Cycle
3. Treatment Validation
Investigate Effects
Investigate Trade-offs
Investigate Requirement
2. Treatment Design
Design Requirements
Investigate Treatments
Design new Treatments

Figure 1.5: The *Engineering Cycle* based on Wieringa [148] and adapted from Johanssen [76]. It is separated into five activities, called *Problem Investigation, Treatment Design, Treatment Validation, Treatment Implementation* and *Implementation Evaluation*. The first of those three activities form the *Design Cycle*, a subset of the engineering cycle which is iterative itself and is often performed several times during research.

Figure 1.5 shows the *Engineering Cycle* as describe by Wieringa [148] and adapted from Johanssen [76]. It is a problem solving process consisting of five activities, starting with the `Problem Investigation`, which tries to answer the question "What phenomenon should be improved and why?", and thus learn more about the problem that should be addressed. This is achieved by answering knowledge questions, which we indicate by the keyword *Investigate* as introduced by Johanssen. The second step, called `Treatment Design`, investigates already available treatments, defines the requirements, and if needed creates a new treatment for the problem context. In Figure 1.5, these design problems are indicated by the *Design* keyword. Third, the `Treatment Validation` is based on several knowledge questions and asks whether the `Treatment Design` created an artifact that addressed the investigated problem from the `Problem Investigation` and brings stakeholders closer to their goals without deploying the created treatment to the actual problem domain's context. These first three steps form a subset of the engineering cycle called `Design Cycle`. Independent of the remaining two steps of the engineering cycle, the design cycle is an iterative process itself that can be performed by researchers several times before continuing with the deployment. Design science research projects are usually restricted to this cycle and do not perform the entire engineering cycle.

Fourth, the `Treatment Implementation` transfers the validated treatment to the problem context and is followed by the `Implementation Evaluation` that investigates the treatment when applied by the stakeholders in the problem context. These steps are part of the engineering cycle. The evaluation can motivate the problem investigation of another engineering cycle closing the loop.

1.3.2 Multiple Case Study

As defined by Creswell, "Case study research involves the study of an issue explored through one or more cases with a bounded system." [35]. Therefore, a case study is a study on a person, groups of people or units [62] in their real-world settings [117] when boundaries between phenomenon and context may not be clearly evident [157]. Yin also distinguishes between holistic and embedded cases studies; while holistic case studies have one case and one unit of analysis per context, embedded case studies can have several units of analysis per case and per context [157]. Each case study goes through three main stages: *Design* and *Plan, Collect, Analyze* as well as *Analyze, Report*. While the Design focuses on the theory development, the second phase focuses on conducting the case study and finally, the last phase focuses on drawing conclusions.

In comparison, multiple case study research "explores [...] multiple bounded systems (cases) over time, through detailed, in depth data collection involving multiple sources of information" [35]. While case studies focus on a singular case, multiple case studies investigate more than one case; leading to more information about the studied phenomenon but with high coupling to the characteristics of the selected cases. If the selected cases provide similar information about the investigated phenomenon, it indicates robustness. Selecting the right cases is therefore crucial for the validity of multiple case studies. The cases should be typical, critical, revelatory or unique in some respect instead of going for availability [14]. Therefore, typical distinctions are the application domain, the application type, some process or the

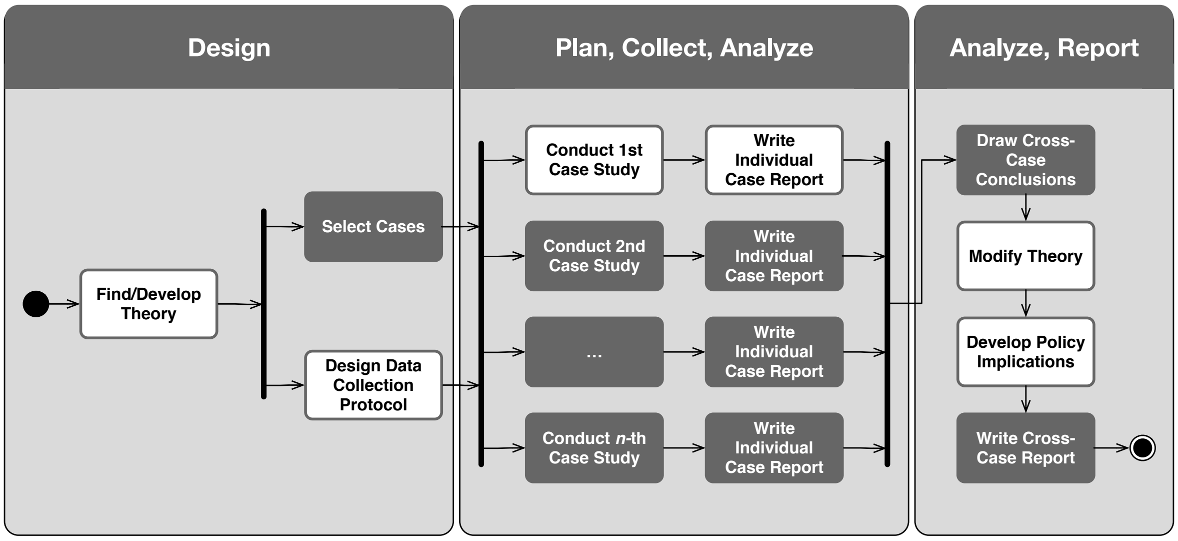


Figure 1.6: Process of Multiple Case Study in an activity diagram adapted from Yin and Runeson [117, 157]. It is separated into three parts, *Design* and *Plan, Collect, Analyze* as well as *Analyze, Report*. The activities in *dark grey* boxes highlight activities that differ from the process of a single case study.

study participants. While the number of cases is coupled with the certainty that needs to be achieved, Yin advises "to settle for two or three literal replications [...] when the issue at hand does not demand an excessive degree of certainty" [157].

The process of a multiple case study is depicted in Figure 1.6 and shows the same three phases as for a regular case study. To highlight the differences between a case study and multiple case study, boxes highlighted in *dark grey* are specific for multiple case study. The first phase, the `Design` is based on three activities. First, `Find/Develop a Theory` which needs to be investigated. Followed by the two steps that can be done in parallel, the `Selection of the Cases` and the `Design Data Collection Protocol` activity. With those activities finished, the `Design` phase is concluded. The first step of the second phase is `Conducting the Case Studies` which is followed by a `Written Individual Case Report` for each case. This is of particular importance for multiple case study, which generates the additional information in this step. The final phase, the `Analyze, Report`, evaluates all the written reports and compares the different cases.

1.4 Research Approach

The research approach of this dissertation is shown in Figure 1.7 and represents one engineering cycle which is rooted in the described problem context.

During the problem investigation, the problem context and the resulting research objectives are investigated in more detail using software engineering techniques as

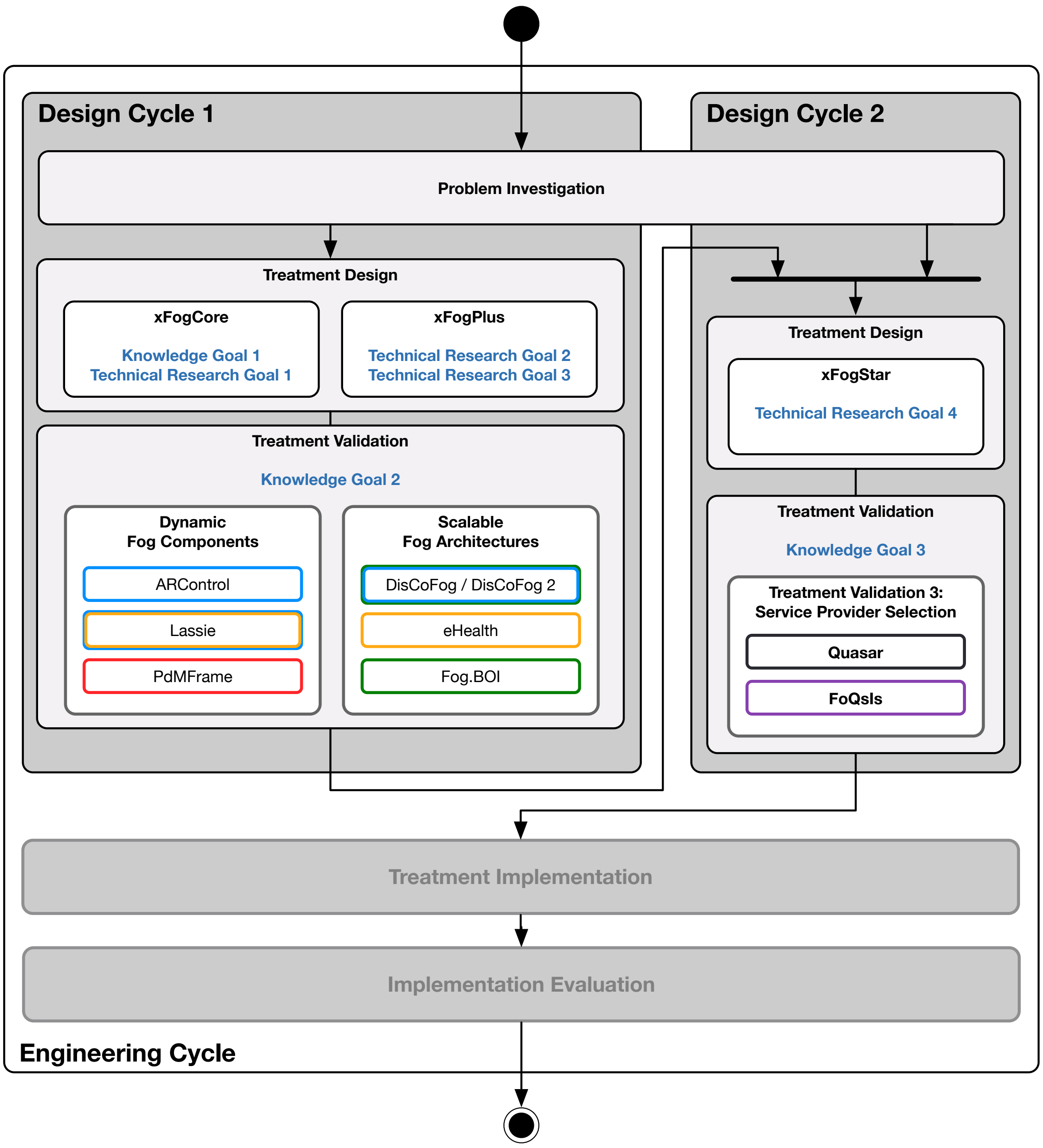


Figure 1.7: The engineering cycle based on Wieringa [148].

described by Bruegge et al. [21]. These led to functional and non-functional requirements that need to be addressed by the xFog framework. Using the design science methodology, the requirements represent *Design Problems* which are derived from the technical research goals.

The problem context as well as the problem investigation triggered two design cycles: The creation of the xFog framework and the design of the xFogStar workflow. Within the first design cycle, the treatment design phase addressed *Knowledge Goal 1* as well as the first three *Technical Research Goals*. This treatment design led to the development of xFogCore and xFogPlus, which are the core components of the xFog framework. The resulting validation investigated *Knowledge Goal 2* to understand the feasibility of the xFog framework. It included two multiple case studies that address *Dynamic Fog Components* and *Scalable Fog Architectures*, which used xFogCore and the two concepts of xFogPlus.

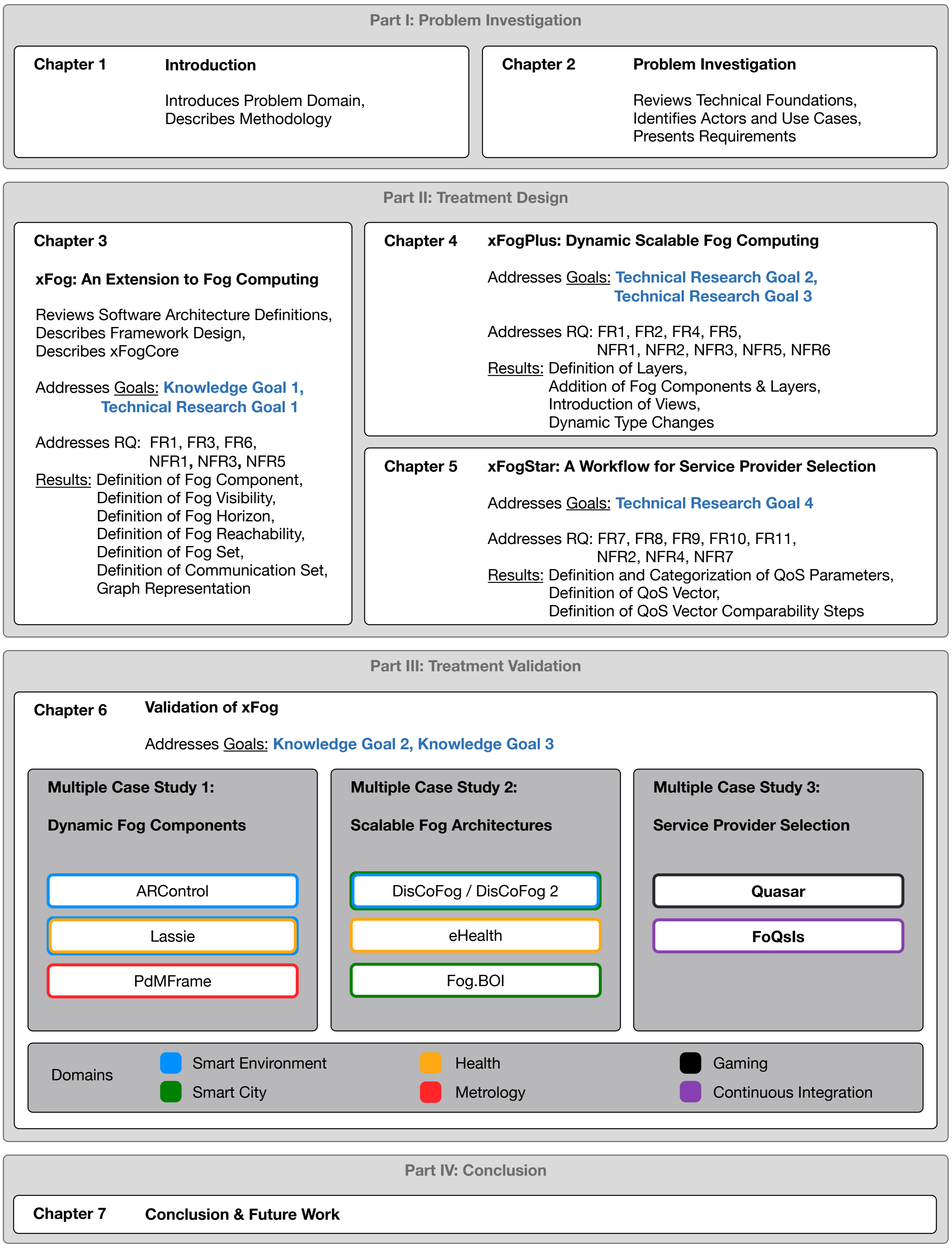
The xFog framework and the problem investigation triggered the second design cycle which resulted in the creation of xFogStar. xFogStar is an artifact developed to address the *Technical Research Goal 4*. In the subsequent validation, we investigated *Knowledge Goal 3* to evaluate the feasibility of the xFogStar workflow. Therefore, we created two instantiations of the workflow.

As for most design science projects, the treatment implementation and implementation evaluation are out of scope for this dissertation and remain future work.

1.5 Dissertation Structure

This dissertation follows an adapted version of the design science methodology by Wieringa [148] as described in Section 1.3.1 and consists of four parts. Figure 1.8 provides an overview of these parts and their internal structure, and connects the individual case studies to the different multiple case studies in the treatment validations. Part I, Part II, and Part III reflect the *Design Cycle* as indicated by their names: *Problem Investigation*, *Treatment Design*, and *Treatment Validation*.

Part I describes and investigates the problem this dissertation addresses and establishes according research goals. In Part II, we describe the xFog framework and xFogStar, which address the first knowledge goal and the technical research goals introduced in Section 1.2. The third part describes the treatment validation including three multiple case studies to investigate the knowledge goals that are concerned with the feasibility of xFog and xFogStar. Finally, Part IV concludes this dissertation.



Part I: Problem Investigation

Chapter 1 **Introduction**

Introduces Problem Domain,
Describes Methodology

Chapter 2 **Problem Investigation**

Reviews Technical Foundations,
Identifies Actors and Use Cases,
Presents Requirements

Part II: Treatment Design

Chapter 3

xFog: An Extension to Fog Computing

Reviews Software Architecture Definitions,
Describes Framework Design,
Describes xFogCore

Addresses Goals: **Knowledge Goal 1,**
Technical Research Goal 1

Addresses RQ: FR1, FR3, FR6,
NFR1, NFR3, NFR5
Results: Definition of Fog Component,
Definition of Fog Visibility,
Definition of Fog Horizon,
Definition of Fog Reachability,
Definition of Fog Set,
Definition of Communication Set,
Graph Representation

Chapter 4 **xFogPlus: Dynamic Scalable Fog Computing**

Addresses Goals: **Technical Research Goal 2,**
Technical Research Goal 3

Addresses RQ: FR1, FR2, FR4, FR5,
NFR1, NFR2, NFR3, NFR5, NFR6
Results: Definition of Layers,
Addition of Fog Components & Layers,
Introduction of Views,
Dynamic Type Changes

Chapter 5 **xFogStar: A Workflow for Service Provider Selection**

Addresses Goals: **Technical Research Goal 4**

Addresses RQ: FR7, FR8, FR9, FR10, FR11,
NFR2, NFR4, NFR7
Results: Definition and Categorization of QoS Parameters,
Definition of QoS Vector,
Definition of QoS Vector Comparability Steps

Part III: Treatment Validation

Chapter 6 **Validation of xFog**

Addresses Goals: **Knowledge Goal 2, Knowledge Goal 3**

Multiple Case Study 1:

Dynamic Fog Components

ARControl
Lassie
PdMFrame

Multiple Case Study 2:

Scalable Fog Architectures

DisCoFog / DisCoFog 2
eHealth
Fog.BOI

Multiple Case Study 3:

Service Provider Selection

Quasar
FoQsIs

Domains
🟦 Smart Environment 🟧 Health ⬛ Gaming
🟩 Smart City 🟥 Metrology 🟪 Continuous Integration

Part IV: Conclusion

Chapter 7 **Conclusion & Future Work**

Figure 1.8: Overview of the dissertations's structure.

Chapter 1 introduces the problem domain of the dissertation, establishes the research goals, and describes the used research methodology of design science and multiple case studies.

Chapter 2 investigates the described problem from Chapter 1 using software engineering techniques as described by Bruegge et al. [21]. We identify actors, use cases, and requirements which form the basis for the xFog framework based on the research goals. We establish the technical background including the concepts of Internet of Things, Fog Computing, and Quality of Service.

Chapter 3 presents the first treatment design to address the investigated problems and to set the foundations for the following treatments. It describes XFOG, an e<u>x</u>tension for <u>**Fog**</u> Computing, to formalize Fog Computing as a software architecture based on sets and graphs. Therefore, xFogCore establishes the concepts *Fog Component*, *Fog Visibility*, *Fog Horizon*, *Fog Reachability*, *Fog Set*, and the Communication Set.

Chapter 4 introduces the treatment design called XFOGPLUS. It adds support for dynamics and scalability which enables components and layers to be dynamically added and removed from a Fog Architecture. Therefor, we establish definitions for the different layers of Fog Computing. We define a *View* concept to support stakeholders to address specific parts of Fog Architectures.

Chapter 5 introduces the third treatment design: XFOGSTAR. It represents a concept that is enabled by xFog and its foundations provided by xFogCore and xFogPlus. xFogStar is a workflow that allows Fog Components within the xFog framework to find the "best" fitting communication partner for a requested service among several service providers in one Fog Horizon. Therefor, the Fog Component uses a discovery mechanism to find all available service providers and quality of service parameters to define its needs and preferences.

Chapter 6 validates the xFog framework and xFogStar using multiple case studies and domain engineering. We address three aspects reflecting the foundations of xFog and the xFogStar workflow: *Dynamic Fog Components*, *Scalable Fog Architectures*, and the *Service Provider Selection*.

Chapter 7 concludes this dissertation by summarizing its contributions and providing an outlook on concepts that can be addressed in the future.

Chapter 2

Problem Investigation

"Software architects should design, develop, nurture, and maintain the architecture of the software-intensive systems they are involved with."

— PHILIPPE KRUCHTEN [82]

The problem investigation tries to accurately describe the phenomenon that should be improved. It does so by investigating the status quo of the problem domain as well as the reasoning why the problem needs improvement. Based on the problem description in Chapter 1, in this chapter, we focus on a detailed requirements analysis of the problem as described by Bruegge et al. [21]. Therefore, in Section 2.1, we start by describing the technical background of the problem domain to establish the status quo. The following section, Section 2.2, uses the description of the problem and the set technical research goals to create a use case model which describes the interactions between the actors of the system and the use cases among each other. Finally, Section 2.3 formalizes the presented use cases into requirements that map to design problems in terms of design science. In the following chapters, we address and relate to them during the creation of the framework.

2.1 Technical Background

In this section, we introduce the technical knowledge, the concepts, as well as definitions to create a common understanding of the problem domain. This represents the status quo of the given concepts which we base our framework on. We address three aspects: The Internet of Things, which is the basis for all components that we work with, Fog Computing itself, and the related term Edge Computing, and finally, quality of service, which we use to address the *Technical Research Goal 4*.

2.1.1 Internet of Things

According to Gubbi et al., the Internet of Things (IOT) is defined as interconnected sensing and actuating devices which share information across platforms using data analytics, ubiquitous and Cloud Computing [60]. Therefore, these devices require computational capabilities and are uniquely addressable, making them "close the gap between the physical and the virtual world"[47] through scalable software systems [138, 151]. According to Atzori et al., this approach can be realized in three different ways: things-oriented, internet-oriented, or semantic-oriented [10]. All of those approaches share a huge amount of provided information that allows "to mine and detect patterns, perform predictions or optimization" [38], but also introduces new business models and challenges due to the increasing interconnectivity [32, 48, 140].

As one of the first IoT devices, a Coke vending machine at Carnegie Mellon University in Pittsburgh, USA, in 1982 [1], IoT rapidly spread to all domains. Especially in day-to-day life, IoT devices found a way in all domains: in smart environments, smart textiles, household appliances, etc. This consumer focused trend led to two subclasses to enable the integration into production lines [92] and to increase operational effectivity [153]: Consumer IoT (CIOT) and Industrial IoT (IIOT) that focuses on IIoT devices which have a bigger emphasis on reliability and connectivity [125]. Traditionally, IIoT systems use centralized architectures that require sensor data, actions and control commands to be routed through the entire network [95], but advances in miniaturization and costs shifted the role of sensors and actuators from simple interaction devices to autonomous systems with general-purpose computers [20, 135, 153].

2.1.2 Fog & Edge Computing

In this section, we address Fog Computing and Edge Computing but also mention other concepts with the same intention. The main goal of all of these concepts is to reduce latency, increase the available context by introducing location awareness, and reduce communication overhead [127]. The problems that should be addressed by these goals can mainly be traced back to the growing interest in the IoT. Already in 2018, a total of 17.8 billion active device connections were calculated of which 7 billion are considered IoT devices. Until 2025, depending on the prognosis, this number is predicted to rise to between 34.2[2] and 75[3] billion devices with more than 60% of them being IoT. This increase results in a vast amount of data which makes networks the bottlenecks and challenges centralized approaches such

[1]http://www.cs.cmu.edu/ coke/

[2]https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

[3]https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

as Cloud Computing. Distributed approaches can not only help to regulate the traffic by aggregating and preprocessing the data [70, 112], but also by making decisions that do not require central instances [52, 93, 127, 128, 129], making the existing architectural boundaries blurry [153]. Due to the technological advancements already addressed in Section 2.1.1 on IoT, the central processing unit (CPU), graphics processing unit (GPU), storage, etc. of devices at the edge increases and should be exploited by offering a wide variety of services [20, 77]. This idea was also addressed by Bonomi et al. in 2012 by bringing the power of central instances closer to the consumer, and therefore the edge that result in lower latencies [17, 18]. While services and data evaluation is pushed closer to the edge, most decentralized services still leverage the benefits of Cloud Computing, by offloading big data analytics or machine learning tasks [19]. Finally, many IoT applications rely on location awareness, as for instance in smart environments as shown by Alletto et al. and Applin et al. [6, 8], which Cloud Computing lacks.

Although the terms Fog Computing and Edge Computing are often used as synonyms, as for instance by Varghese et al. and Vi et al. [143, 155], we differentiate two important aspects. While Edge Computing tries to push services and data as close to the edge as possible, and therefore heavily uses Edge Networks, Fog Computing includes every device on the way from the cloud to the edge [85]. We consider Edge Computing a subset of Fog Computing, which is also supported by Mahmud et al. [91].

Both, Edge Computing and Fog Computing, describe a three layer approach. The cloud represents the topmost layer providing remote services and at the other end, the end user devices, also called Edge Devices, are on the Edge Layer. In both concepts, an intermediate layer is in between those layers which we call Fog Layer.

Figure 2.1 shows an hierarchical overview of this structure. Additionally, it shows the properties of Fog Computing. The closer a device is to the cloud, the more *Resources* it has available, the more *Reliable Connectivity* is ensured, the more *Reliability* it provides, and the greater the *Latency* is to devices at the edge. On the other side, the closer a device is to the edge of the network, the more *Devices* are present on its and adjacent layers, the more important are *Real-Time* requirements, *Dynamic* aspects, *Location Awareness*, *Geo Distribution*, and *Interactivity*. Edge and Fog Computing differ on the placement of this intermediate layer that is used to provide services, lower bandwidth usage and latency, and allows locality [18, 120, 142]. These intermediate layers consist of heterogeneous devices [138] and can be seen as "mini-clouds" [142].

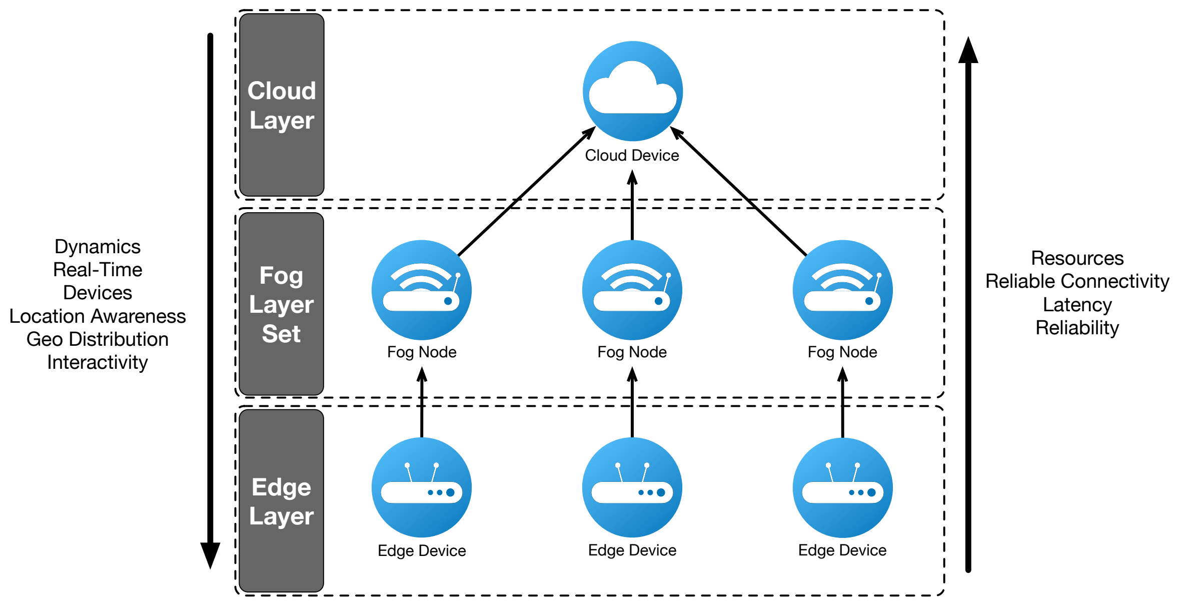


Figure 2.1: Hierarchical structure of Fog Computing adapted from Seitz and Yousefpour [126, 158]. In addition to the structure, it also shows that the available *Resources*, *Reliable Connectivity*, *Latency*, and *Reliability* increase the closer the component is to the *Cloud*. On the other side, the amount of *Devices*, the *Real-time* requirement, *Dynamics*, *Location Awareness*, *Geo Distribution*, and *Interactivity* increase the closer the component is to the *Edge*.

The following definition by Yi et al. [155] reflects not only a description for Fog Computing, but also for Edge Computing:

> *"Fog computing is a geographically distributed computing architecture with a resource pool consists of one or more ubiquitously connected heterogeneous devices (including Edge Devices) at the edge of network and not exclusively seamlessly backed by cloud services, to collaboratively provide elastic computation, storage and communication (and many other new services and tasks) in isolated environments to a large scale of clients in proximity." [155]*

In addition to the mentioned characteristics by Yi et al. [155], Bonomi et al. add the importance of interoperability between heterogenous devices and the support of mobility [18].

2.1.3 Quality of Service

Quality of service (QoS) describes non-functional properties [50], qualitative or quantitative, that define and can be assigned to any kind of service offered or consumed. These properties can also be referred to as resource or service metrics [87, 91, 116]. Especially in computer science, quality of service is typically found in the area of networking, as for instance in [29, 37, 55, 144], real-time applications

[154], or middleware [50, 161]. Depending on the application domain, different non-functional properties are of importance. These application domains specify in which way quality of service is used. While in networking QoS is used to find optimal routing strategies, real-time applications focus on the selection of the "right" service based on different characteristics. Common properties are *Time*, *Cost*, *Data*, and *Energy* [27, 91, 161]. In particular in real-time applications, properties such as *Reliability* and *Availability* are suggested [50]. The emphasis of Fog Computing on real-time also pushes those properties into its QoS. According to Yi et al., quality of service parameters in the fog environment are divided into the four categories *Connectivitity*, *Reliability*, *Capacity*, and *Delay* [156].

After collecting QoS parameters, the properties need to be compared to select the best network route, the cheapest service, or the most available service. This can be achieved by hand, comparing single values as for instance for the cheapest service, or by using extensive calculations, as for example in the QoS Model by Liu et al. [90], Al-Mastri et al. [4], or Zheng et al. [160].

2.2 Use Case Model

Figure 2.2 shows the use case model for an extension for Fog Computing. The use cases are derived from the problem description in Section 1.1, the technical research goals in Section 1.2, and are based on the technical background.

It distinguishes between two types of actors: `Component Developers` and `Software Architects`. While `Component Developers` implement single components and their functionality, and accordingly, care about the direct surrounding of the component, `Software Architects` have the bigger picture of the entire Fog Architecture in mind.

The `Software Architect` wants to `Define Architecture Boundar[ies]` and keep them up to date during the lifetime of the architecture. This involves to know which components, connectors and layers are present. Therefore, the use case `Define Architecture Boundary` includes the three use cases: `Define Components`, `Define Connectors`, and `Define Layers`. These use cases also represent functional requirements based on the definitions of a software architecture which we address in Section 3.1.1. The definition of layers is linked to the offered services on each layer. Accordingly, `Define Layers` includes the use case `Limit Components to Service` which extends `Define Components`.

In addition to defining the boundaries, a `Software Architect` can `Add new Component[s]` and `Add new Layer[s]`. These use cases describe the dynamic properties of Fog Architectures and allow `Software Architects` to integrate new components and layers at runtime.

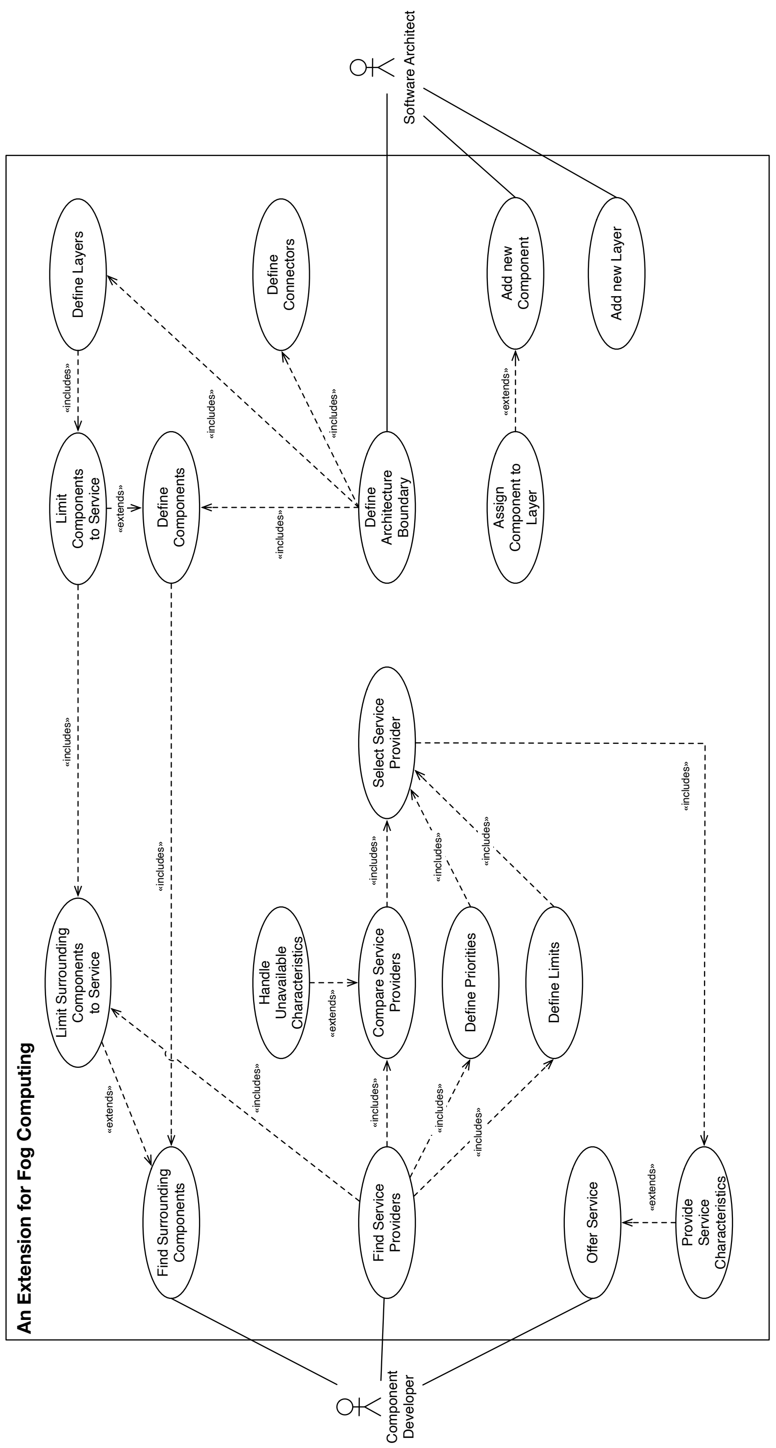

An Extension for Fog Computing
Software Architect
Component Developer
Define Layers
Define Connectors
Add new Component
Add new Layer
Limit Components to Service
Define Components
Define Architecture Boundary
Assign Component to Layer
Select Service Provider
Limit Surrounding Components to Service
Handle Unavailable Characteristics
Compare Service Providers
Define Priorities
Define Limits
Find Surrounding Components
Find Service Providers
Offer Service
Provide Service Characteristics
«includes»
«extends»

Figure 2.2: The use case model for an extension for Fog Computing (UML Use Case Model). It depicts the interactions of the two actors, *Software Architect* and *Component Developer*, with the system. While the *Software Architect* is more concerned with the definition and design of the Fog Architecture, and therefore defining a clear system boundary, the *Component Developer* represents an actor that develops individual components of the architecture. Thus, the *Component Developer* is more interested in smaller sections of the Fog Architecture. These relations are also highlighted by the use cases the actors interact with.

After adding a new component, the `Software Architect` can `Assign [the] Component to [a] Layer` which is an extension to the use case: `Add new Component`.

While a Fog Architecture is designed by one or only a few `Software Architects`, usually several `Component Developer[s]` are included. From their perspective, they need to know who is in their close surrounding, and thus who they can interact with. Accordingly, a `Component Developer` can `Find Surrounding Components` which is related to `Define Components`, that `Software Architects` use to define all involved Fog Components of a Fog Architecture, just on a smaller scale. Additionally, they can `Limit Surrounding Components to [a] Service` which is an extension to `Find Surrounding Components` and is included in `Limit Components to Service`. Limiting all components to a service allows `Component Developers` to find all components in the surrounding that are interested in a specific service.

`Component Developer[s]` can also `Find Service Providers` and `Offer Services`. Finding service providers for a specific service includes `Limit Surrounding Components to Service`. If one out of many service providers should be selected, the `Component Developer` has to `Compare Service Providers`, `Define Priorities` on the service providers characteristics, and `Define Limits` such as a maximum price for the service. If service providers do not provide information, e. g., for the energy consumption of a component, `Component Developers` also have to `Handle Unavailable Parameters` which is an extension to the comparison of service providers. Finally, these steps lead to a `Select[ion of a] Service Provider`. On the other side, `Component Developer[s]` can `Offer Service[s]`. To support the selection process of service providers, the use case can be extended by `Provide Service Characteristics`.

2.3 Requirements

Based on the *Technical Research Goals*, the proposed use cases, and the actors, we derive the functional (FR) and non-functional requirements (NFR) for the design of the framework. These requirements refer to the *Design Problems* of the design science approach which need to be addressed to achieve the *Technical Research Goals*. Section 2.3.1 lists the functional requirements grouped by the two identified actors: *Software Architect* and *Component Developer*.

The requirements based on the needs of the *Software Architect* describe functionality to support the creation of a Fog Architecture, adjust it to changing needs, and maintain it during its lifecycle.

The *Component Developer's* requirements, on the other hand, address the needs that arise during the development process and lifecycle of individual components and their relation to other components within their surrounding.

2.3.1 Functional Requirements

Software Architects have many responsibilities, among others, they have to "[define] the architecture of the software", "[maintain] the architectural integrity of the software", and "[propose] the order and contents of the successive iterations" [81]. We summarize these tasks as the definition of clear architectural boundaries which is especially important in a set of highly connected components that can be dynamically added and removed from the architecture. This dynamic aspect poses a challenge to the maintenance of the architecture's integrity and needs to be addressed. Therefore, we want to enable *Software Architects* that use the extension of Fog Computing to add new components to the architecture and define their used communication channels. These components have to be assigned to the different layers of a Fog Architecture, but it should also be possible to add new layers to enable scalability.

The following four requirements result from these functionalities for an extension for Fog Computing from a *Software Architect's* point of view:

FR1 Define the system boundaries of the Fog Architecture. The system boundary defines which components are part of the architecture and which are not.

FR2 Add new components to the Fog Architecture.

FR3 Define the communication channels which are used by components to connect to other components.

FR4 Assign components to layers.

FR5 Add new layers to the Fog Architecture.

The second actor in the use case model (Figure 2.2) is the *Component Developer*. In comparison to the *Software Architect*, the *Component Developer* is primarily interested in solving the problems of the specific component, they are working on. The solutions to these problems have to fit into the definition of the overarching Fog Architecture, but other components, that the particular component does not interact with, are of less concern. Especially in Fog Computing, with its emphasis on locality, these other components can be found in the immediate surrounding of the component. Therefore, *Component Developers* want to find all components in the surrounding of the developed component. The interactions between components are described as services that can be offered, advertised, and consumed.

When searching for a component that offers a service, it should be possible to restrict the components in the surrounding to those that offer the service. If several potential service providers are remaining, the *Component Developer* should be able to compare the service providers based on their needs. This includes that the *Component Developer* can specify priorities, e. g., preferring energy efficient service providers, but also provide preconditions to characteristics of the service that should not be exceeded, such as the total costs for the service. The name *precondition* is based on the concept "design by contract" by Bertrand Meyer [96]. It allows to specify conditions that need to hold true before a subtask, in our case a service from another component, can be executed. As we do not consider postconditions, we refer to these preconditions as *limits*.

When offering services to other components, *Component Developers* want to provide information about their offered services to the requesting component to ensure that the limits of the requesting component can be met, and therefore increase their chance of being selected.

The respective requirements of the *Component Developer*, and therefore the requirements of the developed component are shown in the following:

FR6 Search for other components in the local surrounding. This search can be restricted to a specific service which a Component Developer needs.

FR7 Compare different service providers based on their characteristics.

FR8 Select a service provider to communicate with.

FR9 Define limits for characteristics of services. Those limits allow, e. g., the specification of a maximum price a component is willing to pay for a service or the minimum provided bandwidth.

FR10 Define priorities to describe which characteristic of a service are of which importance, e. g., putting a high emphasis on sustainability.

FR11 Send the component's service characteristics to other components that request a service from it.

2.3.2 Non-Functional Requirements

This section describes the non-functional requirements for an extension for Fog Computing to describe ″quality attributes and concerns for productivity, time and cost″ [31]. In this context, ″software quality is the degree to which software possesses a desired combination of attributes″ [103]. As proposed by Bruegge et al., we use the FURPS+ model initially proposed by Grady [57] to categorize the non-functional requirements [21]. FURPS+ is an acronym standing for **F**unctionality, which we described in the previous section, and the non-functional categories: **U**sability, **R**eliability, **P**erformance, and **S**upportability. The + indicates the addition of subcategories for each category.

NFR1 Supportability: The Fog Computing extension should allow the addition of new components of different types that are unknown at design time.

NFR2 Supportability: The Fog Computing extension should allow the addition of new layers including services that are unknown during design time.

NFR3 Supportability: The Fog Computing extension should allow the addition of new communication channels and communication media.

NFR4 Supportability: As not all characteristics for the service discovery can be defined for all domains in which Fog Computing can be applied, the list of characteristics should be extensible.

NFR5 Usability: Clear boundaries can be identified at any time of the execution of a Fog Architecture. This allows the identification of components and connectors, as well as the layers the components refer to.

NFR6 Usability: The Fog Computing extension should provide support for large Fog Architectures. These may include more layers than the typical three layers of a Fog Architecture.

NFR7 Reliability: In case a service of a service provider is no longer available, other service providers which offer the same service should be able to substitute the initial service provider.

Chapter 3

xFog: An Extension for Fog Computing

"In my experience, I am sorry to say, industrial software makers tend to react to the system with mixed feelings. On the one hand, they are inclined to think that we have done a kind of model job; on the other hand, they express doubts whether the techniques used are applicable outside the sheltered atmosphere of a University and express the opinion that we were successful only because of the modest scope of the whole project. It is not my intention to underestimate the organizing ability needed to handle a much bigger job, with a lot more people, but I should like to venture the opinion that the larger the project, the more essential the structuring!"

— EDSGER W. DIJKSTRA [40]

xFog is a framework which formalizes Fog Computing and extends it by dynamic and scalable behavior. The dynamic behavior, as introduced in Chapter 1 and investigated in Chapter 2, has several aspects to it which we formalized in functional and non-functional requirements. We use those requirements throughout the framework to show which part of the extension addresses which functionality and needs to fulfill which quality attributes. xFogCore establishes the basic concepts for the extension including the set theoretical approach. It uses sets to describe Fog Computing using the core building blocks of Software Architectures: Components and Connectors. xFogPlus adds the dynamic and scalable aspect to the xFog framework. Finally, xFogStar shows one concept as an addition to xFog which is enabled by the set theoretical concepts: A Workflow for Service Provider Selection. Also we only introduce one workflow, xFogStar shows how xFog creates the basis for many other concepts to come.

Also the entirety of the framework is called xFog, in this chapter, we address xFogCore the core concepts needed for the framework. We address functional requirements FR1, FR3, and FR6, which address functionality that is driven by the definition of a software architecture. These requirements are linked to the non-functional requirements NFR1, NFR4, and NFR5. Thus, in Section 3.1, we take a look at software architectures and how they evolved throughout the history. We introduce the framework's design and explain where the framework's names come from and set the following concepts into context in the meta model:

In Section 3.2, we define the term *Fog Component* which establishes the difference between a component and a component within a Fog Architecture. Based on those Fog Components, we introduce the concepts *Fog Visibility* (Section 3.2.2), *Fog Horizon* (Section 3.2.3), and *Fog Reachability* (Section 3.2.4), which we ultimately use to define the *Fog Set* in Section 3.2.5; the set of Fog Components that defines a Fog Architecture. Additionally, we introduce the idea of limiting the concepts to certain services in Section 3.2.6. We use these as a basis for the following chapters to define dynamics, scalability, and the quality of service based discovery workflow.

The second part of this chapter addresses the *Communication Set* in Section 3.3 and the different communication channels that can be found in Fog Architectures. In Section 3.4, the *Fog Set* and the *Communication Set* are used as the foundation to represent Fog Architectures as graphs.

3.1 Framework Design

This section addresses the design of the framework for Fog Computing as a software architecture. Therefore, we look at the development of software architectures from a historical point of view in Section 3.1.1, compare different software architecture definitions, and highlight their accordances and differences through models to investigate *Knowledge Goal 1*.

In the following Section 3.1.2, we introduce the design of a framework called xFog which is an extension to the Fog Computing concept and focuses on the software architecture implications to achieve the *Technical Research Goal 1*. Additionally, we provide an outlook into the two additional parts xFogPlus and xFogStar, which address the *Technical Research Goal 2*, *Technical Research Goal 3*, and *Technical Research Goal 4*.

3.1.1 Software Architectures

As soon as software systems became more complex and difficult to understand, especially in terms of impact a change might imply, programmers and designers used box-and-line diagrams and written explanations to describe their thoughts [130].

Therefore, using graphs with nodes and edges as a representation technique for software architectures is as old as the discipline itself and has proven to be inherently fitting. While some programmers and designers could identify common approaches and similarities between software system designs, the lack of standardization, best practices, tool support, and the teaching of those, led to low quality software, unmaintainable code, projects running over-budget and time, and often not being delivered at all. This phase, referred to as "software crisis" or "software gap" by attendees of the software engineering conference sponsored by the NATO Science Committee in Garmisch, Germany 1968 [102], had several origins among which the increasing challenges and complexity stood out. This is also highlighted by Dijkstra:

> *"The major cause [of the software crisis] is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming had become an equally gigantic problem."*
>
> — EDSGER W. DIJKSTRA [41]

This crisis motivated several scientists to work on the management of complexity and spreading knowledge within the computer science community. Their key premise was simple: implementation and design are not one combined, but rather two separate activities [15, 115, 145]. This premise introduced but also required the idea of seeing components as black boxes which was based on several ideas such as information hiding as introduced by Parnas [106, 108] to reveal the structure inherent to software systems [107]. Treating components as black boxes allowed to reduce the complexity of software by focusing on structure and design while leaving out implementation details. This encouraged the use of graphs as representations for structure once more. In "On the criteria to be used in decomposing systems in modules" [106] and "The structure of the THE multiprogramming system" [40], Parnas and Dijkstra emphasize the importance of getting the structure right. In retrospective, Dijkstra created in the THE multiprogramming system the first layered architecture.

This focus on structure also enabled the rise of architectural styles — abstractions of common problems in software architecture independent of the problem domain [99]. Several generalized approaches to describe such architectural styles can be found in the work of Mary Shaw, David Garlan, and others [5, 13, 53, 92, 93, 134]. Using these approaches, Garlan and Shaw published several papers refining the their definitions for architectural styles as well as establishing definitions for Software Architecture per se [54, 133, 131, 132, 134] as shown in Figure 3.1.

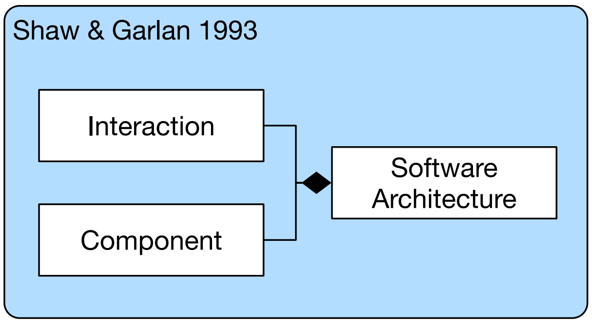


Figure 3.1: The software architecture definition of Shaw and Garlan from 1993 [134] using *Components* and *Interactions* to describe the individual elements (UML class diagram)

For the generalized description of architectural styles, they mainly focused on ″dataflow systems″ such as pipes-and-filters, ″call-and-return systems″ including the hierarchically layered architecture, as presented by Dijkstra [40], ′independent components′ as for instance event systems, ′virtual machines′ like rule-based systems, and ′repositories′ including the blackboard architectural style. This resulted in the following definition:

> *″Software architecture involves the description of elements from which system are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns. In general, a particular system is defined in terms of a collection of components and interaction among those components. Such a system may in turn be used as a (composite) element in a larger system design.″*
>
> — MARY SHAW AND DAVID GARLAN [133]

Perry and Wolf introduced their definition of software architectures in ″Foundations for the study of software architecture″ [109], stating what benefits they try to achieve:

- Architecture is a framework to satisfy requirements.
- Architecture is the technical basis for design, cost estimation, and process management.
- Architecture is the basis for reuse.
- Architecture is the basis for dependency and consistency analysis.

While Shaw and Garlan mainly focused on the components and interactions within an architecture, Perry and Wolf also tried to grasp more informal aspects

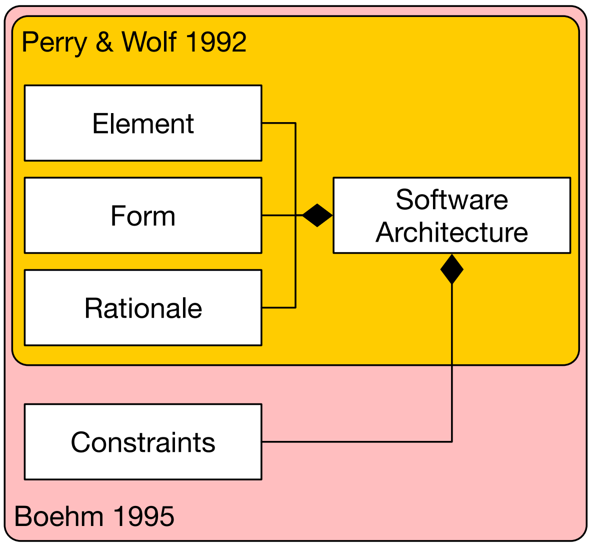


Figure 3.2: Software architecture definition of Perry and Wolf in *yellow* [109] who described architectures based on the three classes *Element*, *Form*, and *Rationale*, and therefore introducing the reasoning behind decisions into architecture. The *red* box shows the extension of Perry and Wolf's definition by Böhm who added *Constraints* [51].

of software architecture such as weights, priorities, properties of elements and rationale, as shown in Figure 3.2. In their definition, components and interactions are summarized under the term `elements` also including data elements, which led to their description of software architecture being a set of elements, their form, and the rationale behind the architecture. Boehm et al. added the idea of constraints shortly after extending the set to four elements [51] as shown in Definition 1.

Definition 1: Software Architecture

Software Architecture = { Elements, Form, Rationale, Constraints }

This model was picked up by Kruchten [80] when describing software architecture with the 4+1 view model. The model describes software architectures as a composition of multiple views depending on the current intent:

- **Logical View:** Represents the object model of the design (End-user, Functionality).
- **Process View:** Represents the concurrency and synchronization aspects of the design (Integrators, Performance, Scalability).
- **Physical View:** Represents the mapping of software onto hardware (System Engineers, Topology, Communications).

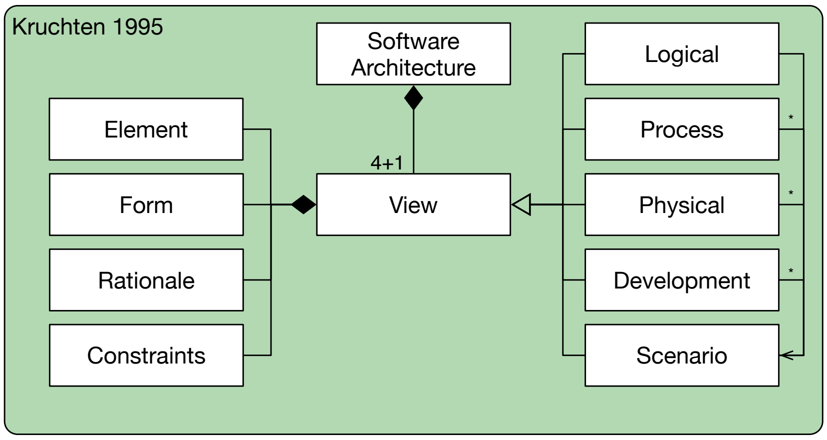


Figure 3.3: Software architecture definition by Kruchten [80] who separated architecture in a collection of 4+1 views shown on the right side and the preexisting architecture components as proposed by Perry, Wolf and Böhm [51, 109] on the left.

- **Development View:** Represents the static organization in its development environment (Programmers, Software Management).

Lastly, the +1 view is an illustration of the four views based on selected use cases or scenarios. The view approach was quickly adapted in the software architecture community and even found its way into more practical reading such as "Applied Software Architecture" by Hofmeister et al. [68]. Hofmeister et al. call their views *Conceptual View*, *Module View*, *Execution View*, and *Code View*. While the conceptual view can directly be mapped to the `Logical View`, module, execution, and code view are subviews of the `Physical view` describing its idea on different granularity levels. Figure 3.3 shows this definition in a compact representation.

In 2013, Bass et al. introduced another distinction [13]. They differentiated between the related terms 'View' and 'Structure'. While views are defined as a representation of architectural elements and the relations among them from a stakeholder perspective, structures are the sets of elements themselves as they exist in software or hardware. Therefore, software architects create structures that are documented for the stakeholder as views. Bass et al. distinguish between three kinds of structures which are shown in Figure 3.4:

- **Module Structures** show a static functional responsibility by separating the system into modules.
- **Component-and-Connector Structures** embody the system as a set of elements (components) and interactions (connectors) with runtime behavior.
- **Allocation Structures** define how the software elements are mapped onto non-software structures such as CPUs, networks, etc.

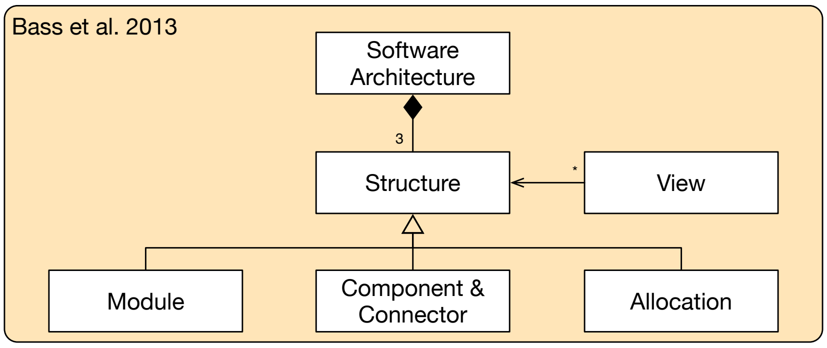


Figure 3.4: The software architecture definition by Bass et al. [13]. They focused on the differentiation between structures and views. While structures are similar to the views, as introduced by Kruchten, Bass et al. argue that structures describe the architecture, whereas views are used to present architecture to stakeholders.

One example representation of the structures can be found in software engineering as shown in [21]. Using the perspective of a software engineer, each structure can be mapped to different UML diagrams describing different software engineering stages. The `Module Structure` is represented as an object design, the `Component-and-Connector Structure` maps to a subsystem decomposition, and the `Allocation Structure` describes a hardware/software mapping.

All software architecture definitions have in common that they build upon basic building components and their connections, disagreeing on the representation of those and the amount of additional information needed to define an architecture. Based on that common idea, it is important to know at any time which components and respectively which connections are part of a software architecture and which are not. This is especially challenging in dynamically changing architectures. This was already addressed by Bass et al. in "Software Architecture in Practice" [13] who introduced the following questions to be answered by architectures:

1. What are the major executing components and how do they interact at runtime?
2. What are the major shared data stores?
3. Which parts of the system are replicated?
4. How does data progress through the system?
5. What parts of the system can run in parallel?
6. Can the system's structure change as it executes and, if so, how?

The first and last question are particularly interesting as they focus on this dynamic idea.

In "The Golden Age of Software Architecture" [130] Shaw and Clements also introduce research fields that still need further investigation:

- Continuing to explore formal relationships between architectural design decisions and quality attributes.
- Finding the right language in which to represent architectures.
- Finding ways to assure conformance between architecture and code.
- Re-thinking our approach to software testing, based on software architecture.
- Organizing architectural knowledge to create reference materials.
- Developing architectural support for systems that dynamically adapt to changes in resources and each user's expectations and preferences.

In particular, we want to address the last research field which is described by Shaw and Clements as follows:

> *"As computing becomes ubiquitous and integrated in everyday devices, both base resources such as bandwidth and information resources such as location-specific data change dynamically. Moreover, each individual user has different needs that change with time. Developing architectures that can dynamically anticipate and react to these changes would help to maximize the benefit each user can obtain. Achieving this will require not only adaptive architectures but also component specifications that reflect variability in user needs as well as intrinsic properties of the component."*
>
> — Mary Shaw and Paul Clements [130]

Based on this description, three important goals can be identified:

1. Developing architectural support for dynamically adapting systems
2. Integrating user's expectations
3. Allowing user preferences

We want to address these goals using Fog Computing, as presented in Section 2.1.2, an architectural style in the context of Cloud Computing that needs to be further investigated in terms of software architectures.

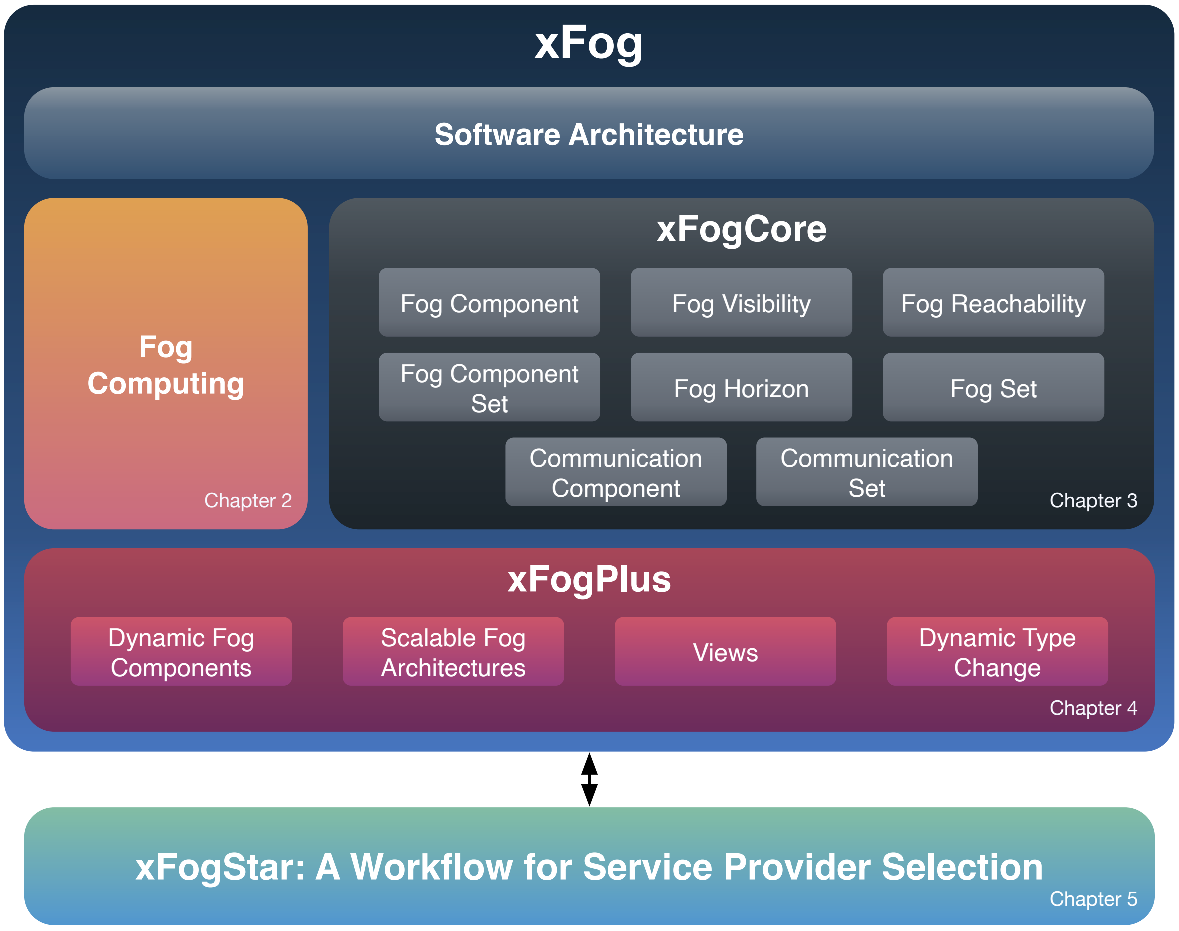


Figure 3.5: Overview of the xFog framework.

3.1.2 xFog Design

The framework xFog addresses the concerns stated by Shaw and Clements that components need to be specified and their intrinsic properties need to be described for adaptive architectures [130]. The name XFOG is short for an **Ex**tension for **Fog** Computing. Its name is inspired by the naming of the extension of the file allocation table, EXFAT for short, which is an adaption of the file allocation table filesystem (FAT) for flash drives [4].

Figure 3.5 provides an overview of the xFog framework. For each of its parts, we designed a logo shown in Figure 3.6 .

xFogCore is an extension for Fog Computing to include the definitions of software architecture which are described by Shaw and Garlan [109]. In Chapter 3, we focus on the definition of the involved components in Fog Architectures, their interactions, and connections to each other in terms of graphs.

[4]https://docs.microsoft.com/en-us/windows/win32/fileio/exfat-specification

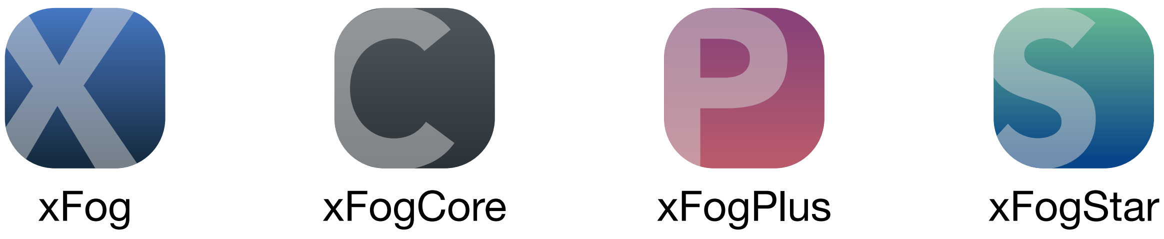


Figure 3.6: xFog framework logos.

The second part of xFog, addressed in Chapter 4, is called XFOGPLUS. It builds upon xFogCore and integrates dynamic and scalable behavior for Fog Architectures: New components can be dynamically added, new layers can be established based on existing components, and entirely new layer can be introduced. Accordingly, the *Plus* in xFogPlus represents the addition of components and layers making the approach dynamic and scalable. xFogPlus defines different Views on Fog Architectures that allow detailed descriptions of the vaguely defined *Fog* within Fog Computing. Based on these Views, components dynamically change their type based on the current level of abstraction between Cloud Devices, Fog Nodes, and Edge Devices.

Using xFog, we introduce a workflow to determine which component within the Fog Architecture should be selected by another component when requesting a service in its surrounding. This workflow is based an QoS parameters and called xFogStar. xFogStar is presented in Chapter 5. The *Star* in its name refers to the idea of the modeling language i*[5] (pronounced i star) which tries to answer the questions *WHO* and *WHY* rather than *WHAT* and is therefore actor-oriented and goal-oriented. xFogStar describes *WHO* should communicate with *WHOM* based on different characteristics which specify the *WHY*.

The remaining of this chapter addresses the concepts of xFogCore. Figure 3.7 provides an overview of the concepts on the meta model level M2. A short description of MOF can be found in Section 3.2.1. The meta model can be used as a basis for applications to understand the involved components, the used communications sets, and the context in which the application will exist. Thus, it provides an initial understanding of the approached Fog Architecture using UML notation.

Based on the definition of Shaw and Garlan [134], each *Software Architecture* consists of multiple *Components* and *Connectors*. A `Fog Architecture`, as a subclass of a `Software Architecture`, consists of one `Fog Set` and one `Communication Set`. While the `Fog Set` relates to the components of the `Software Architecture`, the `Communication Set` relates to the connections between them. The `Fog Set` is based on the set definition introduced in

[5]https://en.wikipedia.org/wiki/I*

Section 3.2.5. Thus, each `Fog Set` consists of multiple `Fog Horizons` that are transitively connected to each other, creating a unique structure as shown in Proof 1. Each `Fog Horizon` is defined by two `Fog Visibilities` that form a symmetric pair. In contrast, the `Fog Reachability`, also consisting of `Fog Visibilities`, is the set that includes the transitive closure but lacks the symmetric property. The `Fog Visibility` itself is defined by its `Fog Components` which in turn are part of the `Fog Component Set` as shown in Section 3.2.7. As describe in Section 3.2.1, the `Fog Component` is of type `Fog Type` allowing it to be considered as a Cloud Device, a Fog Node, or an Edge Device depending on the perspective of the viewer. These *Localities*, as described by Seitz in [126], provide the basis of each Fog Architecture.

Each `Fog Component` can *Provide*, *Consume*, or have *Interest* in several `Services` which in turn can be used to reduce the five sets `Fog Component Set`, `Fog Visibility`, `Fog Horizon`, `Fog Reachability`, and `Fog Set` to all `Fog Components` that are involved with the specific `Service`.

The `Communication Set`, on the other hand, consists of `Communication Components`. `Communication Components` form a triple of two connected `Fog Components` over one `Communication Channel`. These `Communication Channels` can be unidirectional or bidirectional, as described in Section 3.3.

3.2 Component Set

In the following sections, we define different concepts which we ultimately use to define the set of components that compose a Fog Architecture. These concepts are established on set theory, a mathematical discipline that was defined by Georg Cantor in [24] in 1984, creating sets, related relations, and comparisons between those sets. This allows us to describe which elements are considered part of a Fog Architecture, and which are not, providing the mathematical formalizations for Fog Computing in terms of software architectures. We already presented the ideas for these sets in our paper "Fog Horizons–A Theoretical Concept to Enable Dynamic Fog Architectures" [66]. In the following, we present these initial ideas with adjustments and extensions. Beginning in Section 3.2.1, we establish the idea of a Fog Component which we will use in the remaining sections as the basic building blocks, the presented sets are made of.

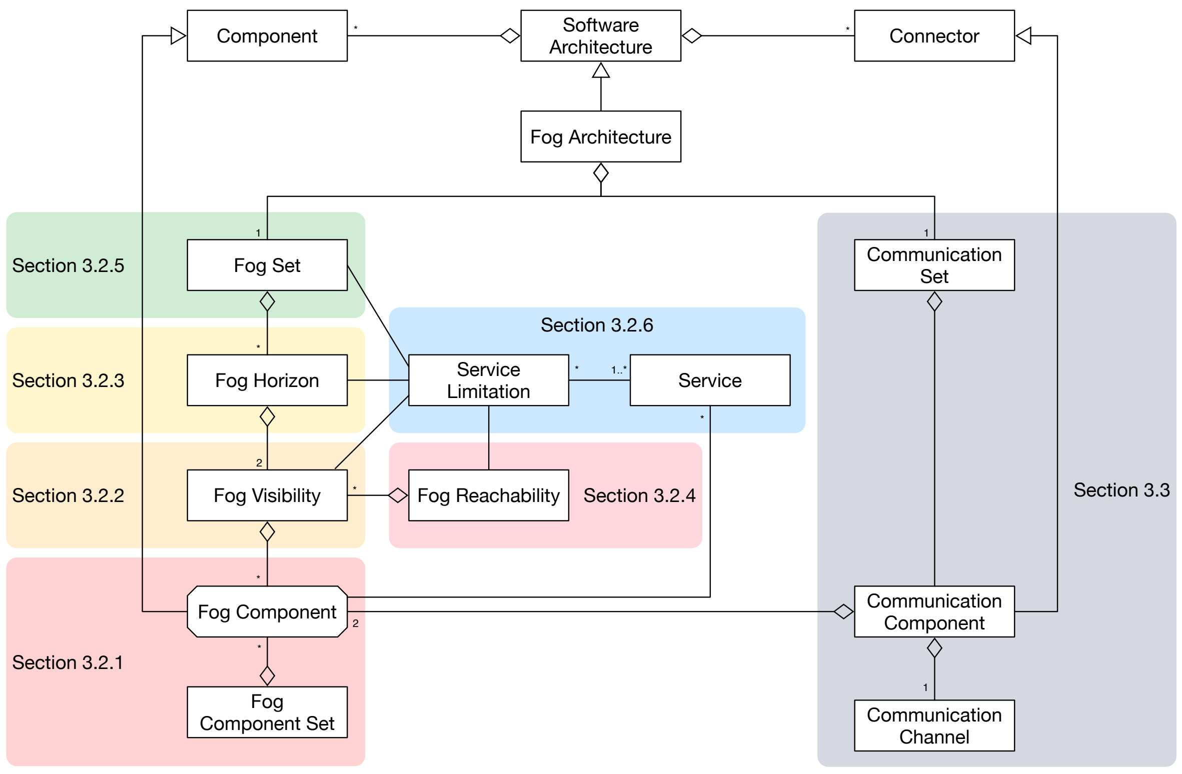


Figure 3.7: Dynamic, scalable Fog Architecture meta model on MOF level M2 setting the set concepts introduced in Chapter 3 in the context of the *Fog Architecture*. Each section with the corresponding concept is highlighted in different colors. The *Fog Architecture* is defined as a subclass of *Software Architecture* and introduces the equivalents to *Components* and *Connectors* as defined by Shaw and Garlan in [134].

3.2.1 Fog Component

As described in Section 2.1.2, Fog Computing introduces three types of devices: *Cloud Devices*, *Fog Nodes* and *Edge Devices*. The Cloud Devices are a kind of server / data center in the internet which offers storage, computational power, or specific software. The Fog Node, in comparison, has less storage and less computational power available, but is placed on the way between the Cloud and the Edge Devices, which provides faster response times, location awareness, and mobility and can already perform basic operations [18]. Fog Nodes are heterogeneous and can be anything ranging from a Raspberry Pis to drones. Finally, the Edge Devices represent the end-user devices that want to use specific services offered by the Fog Nodes and/or the Cloud Devices.

In order to make the following concepts more understandable without differentiating between the different device types, we define a *Fog Type* as the common superclass for the Cloud Devices, Fog Nodes and Edge Devices on the Meta Object Facility (MOF) level M3, allowing it to be used in meta models on MOF level M2. This is similar to the *Target Matter* definition by Seitz on M2 which could be used

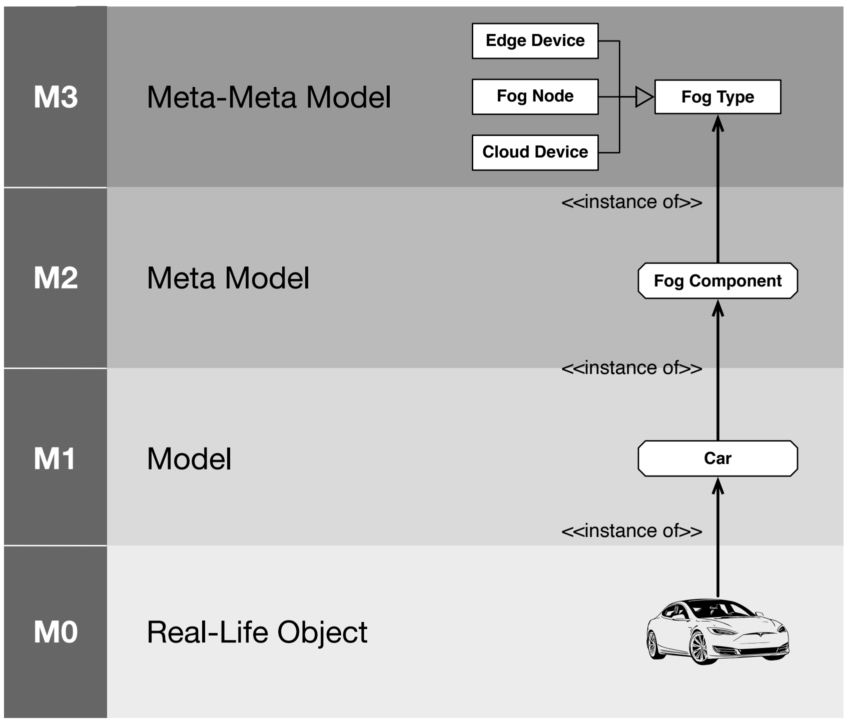


Figure 3.8: Meta Object Facility definition for the four layer approach [58]. Additionally, it shows an example for each layer using the *Fog Type* definition as introduced in Section 3.2.1 on M3 and the definition of a *Fog Component* on M2.

on level M1 [126]. In this context, Edge Devices map to the *Field* stereotype, the Fog Nodes to the *Fog*, and the Cloud Devices to the *Remote* stereotype.

The Meta Object Facility, defined by the Object Management Group, is usually displayed as a four layered approach to describe meta models and their instantiations [58]. It is designed to be an extensible framework, integrating and managing meta-information independent of language, also formerly being extracted from UML [23, 34, 58]. In MOF, each element in one specific layer is defined by the corresponding meta class of the next higher level, thus, being an instance of that meta class. The commonly used four layer framework is shown in Figure 3.8 including the *Fog Type* and *Fog Component* definition down to a concrete example.

Starting at the very bottom, MOF level M0 shows objects of the real life as for example a car, Raspberry Pi, or lightbulb. Those real life objects can be abstracted as part of a model on MOF level M1, as for instance, a UML class or object diagram describing an application. Thus, the real life objects are instances of the model described on the M1 level. The definition of a class or object diagram used on M1 can be found on M2 describing classes, instances etc. On MOF M3, MOF itself is described. Even though this four layer approach is usually connected with the Meta Object Facility, MOF allows an infinite number of layers [58].

Defining a common superclass allows, in addition to reduced complexity, the dynamic behavior of components which we will take a closer look at in Chapter 4.

All instances of the `Fog Type` meta class are called `Fog Components` and form the basis for all concepts which we will introduce in the following. In order to differentiate objects of the `Fog Type` from regular classes without permanently stating the corresponding meta class, we draw classes based on that type in bevelled rectangles.

The first set that we introduce is the *Fog Component Set*. It includes, as shown in Definition 2, all potential Fog Components that we can observe without being tied to any specific Fog Architecture.

Definition 2: Fog Component Set

$\mathsf{FogComponentSet} := \{x \mid x \text{ is a Fog Component}\}$

3.2.2 Fog Visibility

The *Fog Visibility* is the virtual area which can be ”seen” by a Fog Component, giving the concept its name.

Definition 3: Fog Visibility

$\mathsf{FogVisibility}(x) := \{y \mid y \text{ receives direct messages from } x\}$

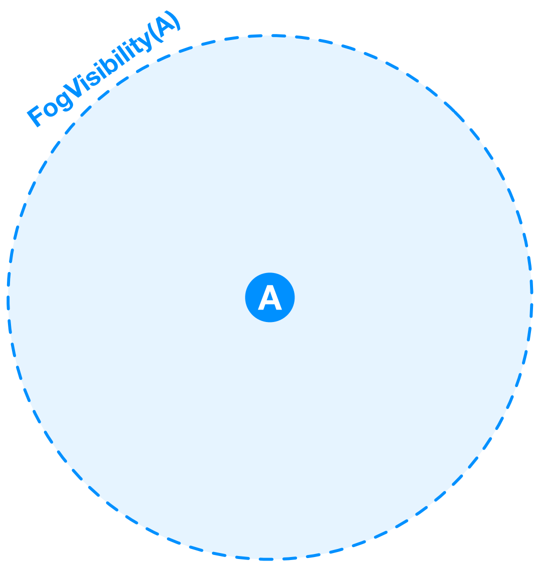
with:
$x, y \in \mathsf{FogComponentSet}$

As shown in Definition 3, the Fog Visibility of a Fog Component x is defined as all potential Fog Components y which can receive direct messages from x. It forms a relation on the superset of all Fog Components allowing us to assign and use relation properties as well as set properties to further specify the concept.

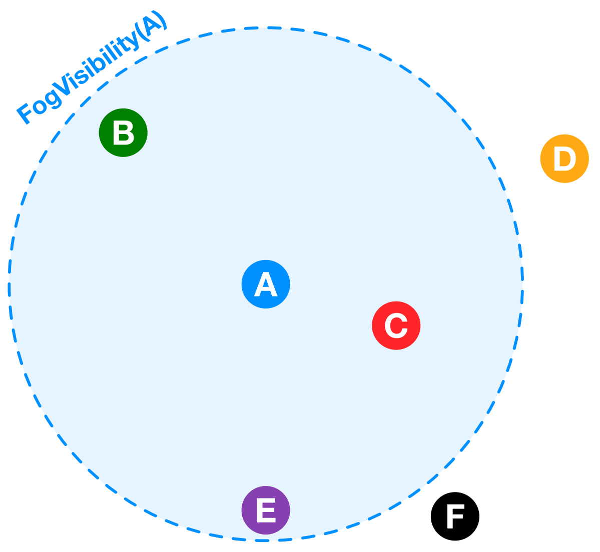
The Fog Visibility’s ”field of view” is limited based on the communication channels used by the Fog Component. While common communication channels such as WIFI, 3G, 4G, or radio frequencies can potentially include a large area, others as for instance Bluetooth or cables might be more limited. We will take a closer look at the Communication Set in Section 3.3, only focusing on the bare minimum which is needed for the understandability of the Fog Set concepts in the following.

One of these aspects is the differentiation of unidirectional and bidirectional communication channels. While bidirectional communication channels allow message exchange in both directions, unidirectional communication channels only allow messages to be sent from one communication partner to the other but not vise versa. This idea is integrated within the Fog Visibility by only requiring messages being exchanged in one direction, from x to y; thus, including all potential communication setups. Another aspect, which supports the more generalized, abstract concept of the Fog Visibility, is the communication channel independency. It is possible, that one Fog Component receives messages from the Fog Component in question over one communication channel, e. g. WIFI, and another Fog Component receiving messages over Bluetooth. While this simplification supports the understandability of the concept, it hides the complexity which occurs when a single Fog Component supports different communication channels. Therefore, the concept define a terminology across communication channel boundaries which leads to a generalized concept.

Also the implementation problems of using different communication channels are not part of this dissertation, the meta model shown in Section 3.1 shows a hierarchy of communication channels which can be exchanged using the strategy pattern. For instance, smartphones switch from 3G or 4G to WIFI as soon as a connection could be established to reduce the data consumption on the mobile contract.



(a) Empty Set.



(b) Set Example adapted from [66].

Figure 3.9: The figure shows two subfigures; both display the Fog Visibility for a Fog Component A displayed as blue circles around the Fog Component A which is displayed as a dot in the middle of the circle. The first figure shows the *Empty Set* with no other Fog Component in proximity except Fog Component A itself. The second figure describes a concrete Fog Visibility example with Fog Components B, C, and E in Fog Component A's Fog Visibility and Fog Component D and F outside.

Figure 3.9 shows the two potential cases of Fog Visibilities for a given Fog Component A – the Fog Visibility as a self containing set and a more general example. While the actual range of the Fog Visibility might differ substantially based on the communication channels used, in this example, we display the range in which a Fog

Component can send messages as circles. All Fog Components within this circle can receive direct messages from the given Fog Component, the Fog Components outside of the circle cannot. While the first example *(a) Empty Set* has limited application use in Fog Architectures, it is interesting from a theoretical point of view showcasing the edge case of the Fog Visibility definition and answers the question if Fog Components without any communication partners still maintain Fog Visibilities. Based on the definition in the Fog Horizon paper [66], the Fog Component's Fog Visibility would be an empty set, while in contrast, the updated definition as presented in Definition 3, includes the Fog Component itself as a viable communication partner as Fog Components could receive direct messages from themselves. This changes the Fog Visibility to be reflexive which we will use in the following definitions.

The second example, while also using the simplified depiction of the Fog Visibility as a circle, is more representative with several Fog Components (B - F) in sight. As shown in Equation 1, the resulting Fog Visibility of Fog Component A is the set of Fog Components A, B, C, and E excluding as shown in Equation 2 D and F which are outside of the circle.

Equation 1: Fog Visibility Example based on Figure 3.9

$$\mathsf{FogVisibility}(A) = \{A, B, C, E\}$$

Equation 2: Fog Visibility Example (2) based on Figure 3.9

$$D,\ F \notin \mathsf{FogVisibility}(A)$$

3.2.3 Fog Horizon

The name Fog Horizon is inspired by two sources. First, it is based on the line which separates the earth from the sky which creates a circular plain of vision for the observer. This idea is also described by the Fog Visibility in the previous section. This circular plain is something that the viewer is familiar with, the area that they know and with which they closely interact. The same idea holds true for the Fog Horizon, devices within the Fog Horizon closely interact with each other, share a common network as for instance a LAN, and also have a locality attached to them.

Second, the term Fog Horizon is based on the concept of event horizons found in physics in the context of black holes. It describes the point, from which nothing can escape the gravitational field of a black hole [141]. This can be seen in the context of Fog Architectures as the virtual area in which a device can interact with others.

In the context of Fog Architectures, as shown in Definition 4, the Fog Horizon is defined as the symmetrical closure of the Fog Visibility. Thus, the Fog Horizon of a given Fog Component x contains all Fog Components y which can send and receive messages to/from Fog Component x, establishing bidirectional communication. Therefore, the Fog Horizon relation is reflexive, due to the definition of the Fog Visibility, and symmetrical. If Fog Component y is in the Fog Horizon of x, Fog Component x is also in the Fog Horizon of y.

Definition 4: Fog Horizon

$\mathsf{FogHorizon}(x) := \mathsf{FogVisibility}(x)^{\leftrightarrow} =$
$\mathsf{FogVisibility}(x) \cap \mathsf{FogVisibility}(x)^{-} =$
$\{y \mid y \in \mathsf{FogVisibility}(x) \ \wedge \ x \in \mathsf{FogVisibility}(y)\}$

with:
$x, y \in \mathsf{FogComponentSet}$

Figure 3.10 shows an example for Fog Horizons. Using the same simplification as in the Fog Visibility example (Figure 3.9), the Fog Visibilities are displayed as circles.

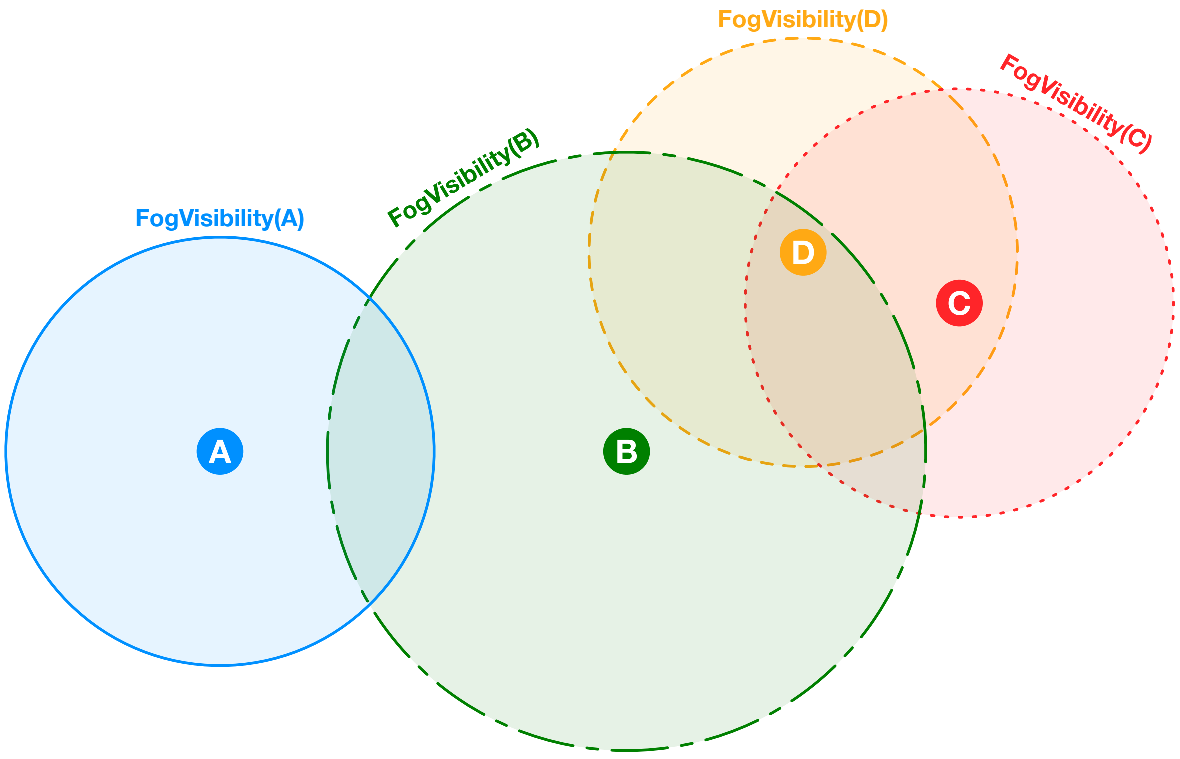


Figure 3.10: Visual example adapted from Henze et al. describing the idea of Fog Horizons with four different Fog Components A, B, C, and D [66]. The according **Fog Visibilities** are represented in different colored circles.

The following Equation 3 lists the Fog Visibilities as well as the Fog Horizons for all Fog Components within the example, showing the different possibilities for Fog Horizons.

The first possibility is shown by Fog Component A: It is an isolated Fog Component not having any other Fog Components within its Fog Visibility, and therefore its Fog Horizon. Fog Component B shows the case in which one Fog Component is in the Fog Visibility of another but not vise versa, breaking the symmetry for the Fog Horizon. However, the Fog Horizons are identical, only containing the Fog Components themselves. Finally, the Fog Components C, D represent the third case in which the Fog Components can send messages to each other; thus, adding each other to the corresponding Fog Horizons.

Equation 3: Fog Horizon Example

$$\begin{aligned}
&\mathsf{FogVisibility}(A) = \{A\}\\
&\mathsf{FogVisibility}(B) = \{B, D\}\\
&\mathsf{FogVisibility}(C) = \{C, D\}\\
&\mathsf{FogVisibility}(D) = \{C, D\}\\
\\
&\mathsf{FogHorizon}(A) = \{A\}\\
&\mathsf{FogHorizon}(B) = \{B\}\\
&\mathsf{FogHorizon}(C) = \{C, D\}\\
&\mathsf{FogHorizon}(D) = \{C, D\}
\end{aligned}$$

3.2.4 Fog Reachability

Using only the Fog Visibility and Fog Horizon concepts, we will not be able to describe an entire set of Fog Components of a Fog Architecture. Doing so, would result in small, isolated sets without any connections between them and no access to any central server. We are missing Fog Components which can be reached using other Fog Components as hops. Thus, the Fog Reachability is, as shown in Definition 5, defined as the transitive closure of the Fog Visibility. The transitive closure adds all components to the set which can be reached through another Fog Component as shown in Figure 3.11. In this figure, the Fog Visibility of Fog Component A contains Fog Component B and the Fog Visibility of Fog Component B contains Fog Component C, allowing Fog Component C to receive messages from Fog Component A. This allows us to not only include components which can send direct messages to a given Fog Component, but also indirect messages forwarded by several other Fog Components.

Definition 5: Fog Reachability

$\mathsf{FogReachability}(x) := \mathsf{FogVisibility}^{+}(x) =$
$\{y \mid y$ receives direct or indirect messages from $x\}$

with:
$x, y \in \mathsf{FogComponentSet}$

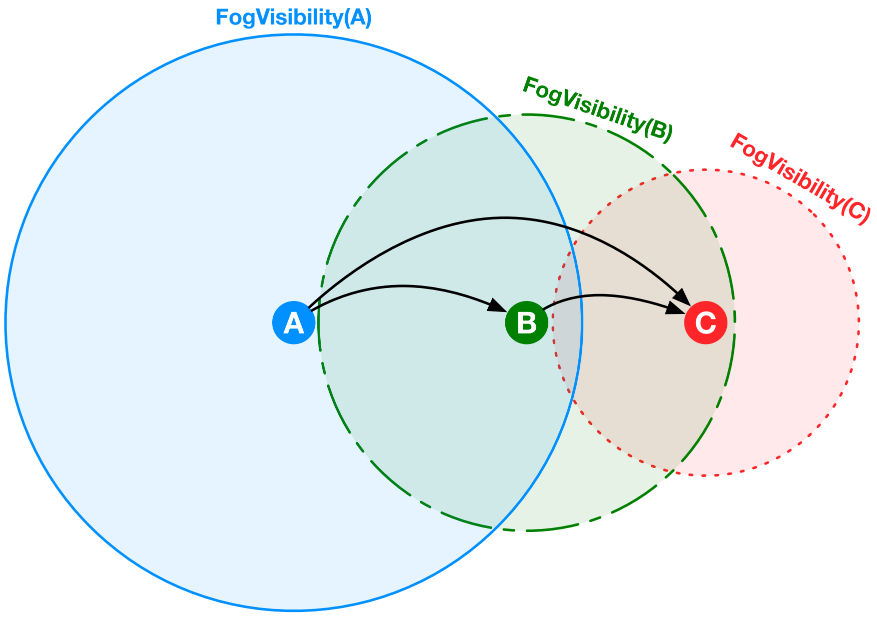


Figure 3.11: The diagram visually depicts the transitive idea of multiple Fog Visibilities. The given example shows Fog Component A reaching Fog Component B and B reaching C. Therefore, A is able to indirectly reach C, while C is not included in its own Fog Visibility.

Figure 3.12 shows an example of four Fog Components A, B, C and D including their Fog Visibilities. The respective Equation 4 shows the Fog Reachabilities for the Fog Components. As Fog Component A's Fog Visibility includes Fog Component B and Fog Component B's Fog Visibility includes the remaining two Fog Components C and D, Fog Component A's Fog Reachability includes the entire set of available Fog Components. Meanwhile, Fog Component A is not included in any Fog Visibility. This prevents the other Fog Components to send messages to Fog Component A excluding it from their Fog Reachability. For Fog Component C with an empty Fog Visibility, the Fog Reachability and Fog Visibility line up only including the Fog Component itself.

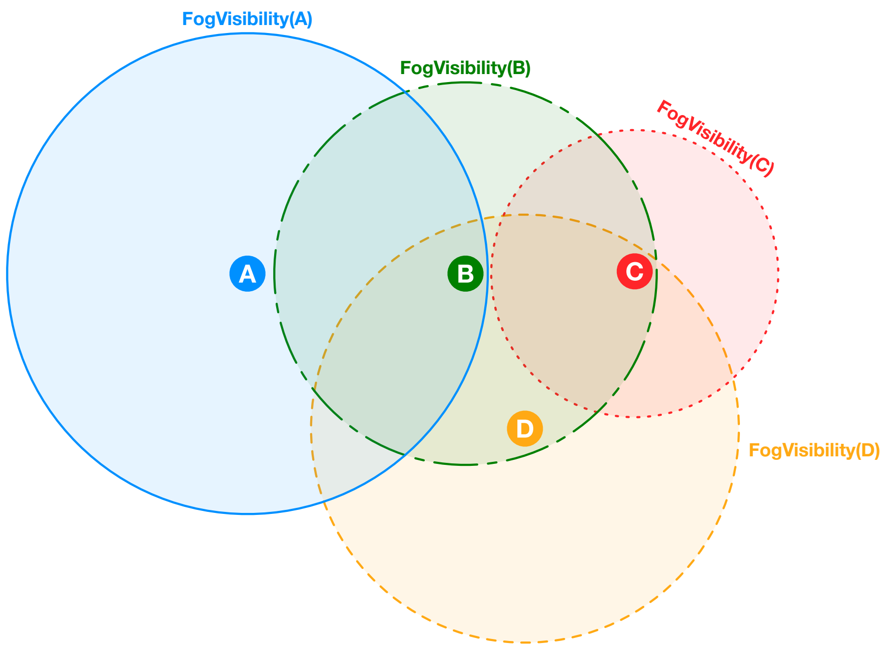


Figure 3.12: The diagram shows an example setup which we will use to present the Fog Reachabilities of the Fog Components A, B, C & D. The different Fog Visibilities of the Fog Components are indicated as different colored circles.

Equation 4: Fog Reachability Example

$$\mathtt{FogReachability}(A) = \{A, B, C, D\}$$
$$\mathtt{FogReachability}(B) = \{B, C, D\}$$
$$\mathtt{FogReachability}(C) = \{C\}$$
$$\mathtt{FogReachability}(D) = \{B, C, D\}$$

The negative side effect of the Fog Reachability shows up as soon as the Fog Architecture is not an isolated system. Whenever an open connection to the internet is involved, there is a potential for the entire internet to be seen as part of the Fog Reachability.

3.2.5 Fog Set

As the Fog Reachability has the potential to include the entire internet based on the architecture's limitations, we would not consider all those Fog Components to be part of a single Fog Architecture. Thus, it is obvious that the Fog Reachability must be a superset of the set of Fog Components which form a Fog Architecture which we call the Fog Set.

Using the same idea as for the Fog Reachability, we can use the transitive closure of the Fog Horizon, as shown in Definition 6, to further reduce the amount of components redefining the idea of the Fog Set. As already shown in Section 3.2.3, Fog Components within a single Fog Horizon interact more closely and can detect each other, supporting the idea of this definition even further.

Definition 6: Fog Set

$$\begin{aligned}&\mathit{FogSet}(x) := \mathit{FogHorizon}^{+}(x) = \\ &\quad \{y \mid y \in \mathit{FogVisibility}^{+}(x) \;\wedge\; x \in \mathit{FogVisibility}^{+}(y)\}\end{aligned}$$

with:

$$x, y \in \mathit{FogComponentSet}$$

Figure 3.13 shows the same Fog Component as we used in the Fog Reachability example, but with greater Fog Visibilities to establish Fog Horizons between connected Fog Components. Equation 5 displays the according Fog Sets for each Fog Component.

Equation 5: Fog Set Example

$$\begin{aligned}&\mathit{FogSet}(A) = \{A, B, C, D\}\\ &\mathit{FogSet}(B) = \{A, B, C, D\}\\ &\mathit{FogSet}(C) = \{A, B, C, D\}\\ &\mathit{FogSet}(D) = \{A, B, C, D\}\end{aligned}$$

As shown in the example, all Fog Sets for the different Fog Components A, B, C, and D are identical, which raises the question: Is the concept of a Fog Set identical for a connected set of Fog Components, and therefore unique?

To answer this question, the following Table 3.1 shows the properties of the individual concepts.

The changed definition of the Fog Visibility including the requesting Fog Component creates the reflexive closure, in comparison to the definition provided in the Fog Horizon paper [66]. The Fog Horizon is symmetrical based on its definition as the symmetrical closure of the Fog Visibility.

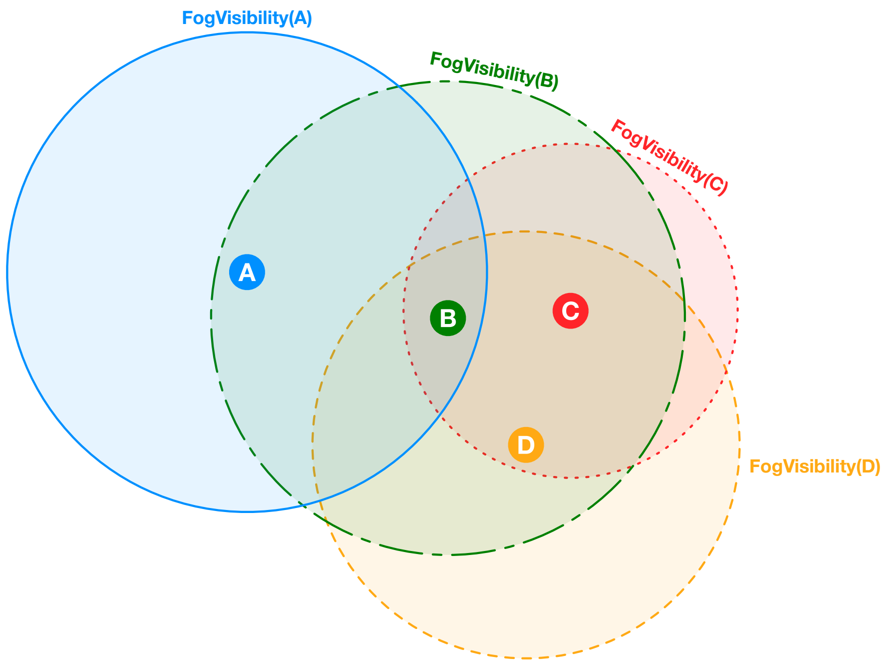


Figure 3.13: Adapted version of Figure 3.12 which includes overlaps that introduce Fog Horizons instead of Fog Visibilities.

Table 3.1: The properties of the different introduced sets based on group theory.

	Reflexiv	Symmetric	Transitiv
FogVisibility	X		
FogHorizon	X	X	
FogReachability	X		X
FogSet	X	X	X

With those two properties and the Fog Set being the transitive closure of the Fog Horizon, the Fog Set relation is an equivalence relation. This means that the Fog Set relation partitions a given Fog Component set in disjunct subsets. Therefore, the Fog Set is unique for all Fog Components in each subset creating an equivalence class as presented in Definition 7.

Definition 7: Uniqueness

In comparison to the definition of the *Fog Visibility* and *Fog Horizon* as presented in Section 3.2.2, respectively Section 3.2.3, which are dependent on the component they are evaluated for, the *Fog Set* is unique for all components within those sets—thus being identical creating equivalence classes:

$$\text{Consider } x, y \in \mathsf{FogComponentSet} \text{ and } x, y \in \text{same Fog Set:}$$
$$\Rightarrow \mathsf{FogSet}(X) = \mathsf{FogSet}(Y)$$

The following Proof 1 shows the correctness of Definition 7, and thus the equality of the two Fog Sets using the properties of equivalence relations. This allows us to omit the parameter for which we are creating the Fog Sets, as long as there is only a single Fog Set involved. As soon as an architecture encapsulates several independent Fog Sets, the provided parameter uniquely identifies each Fog Set, also each Fog Component within each Fog Set results in the same set, as proven before.

Proof 1: Fog Set Uniqueness

Consider $A \subseteq \mathit{FogComponentSet}$ and $x,\ y \in A$:

To show:
Either the Fog Sets of x and y do not have any overlapping components or the Fog Sets are equal:

$$\mathit{FogSet}(x) \cap \mathit{FogSet}(y) = \emptyset \tag{3.1}$$
$$\mathit{FogSet}(x) = \mathit{FogSet}(y) \tag{3.2}$$

Proof by contradiction:
1. Case: If 3.1 holds true, 3.2 cannot hold:

$$\emptyset = \mathit{FogSet}(x) \cap \mathit{FogSet}(y) \tag{3.3}$$
$$\overset{\mathit{Using}3.2}{=} \mathit{FogSet}(x) \cap \mathit{FogSet}(x) \tag{3.4}$$
$$= \mathit{FogSet}(x) \not\Rightarrow \mathit{Reflexivity} \tag{3.5}$$

2. Case: If 3.1 does not hold true, 3.2 has to hold true:
Assumption: It exists a combination of x and y for which the corresponding Fog Sets' intersection is not empty.

$$\exists\, z \in A : z \in \mathit{FogSet}(x) \wedge z \in \mathit{FogSet}(y) \tag{3.6}$$
$$\overset{\mathit{Symmetry}}{\Longrightarrow} x \in \mathit{FogSet}(z) \wedge z \in \mathit{FogSet}(y) \tag{3.7}$$
$$\overset{\mathit{Transitivity}}{\Longrightarrow} x \in \mathit{FogSet}(y) \tag{3.8}$$
$$\Longrightarrow \mathit{FogSet}(x) \subseteq \mathit{FogSet}(y) \tag{3.9}$$

$$\exists\, z \in A : z \in \mathit{FogSet}(x) \wedge z \in \mathit{FogSet}(y) \tag{3.10}$$
$$\overset{\mathit{Symmetry}}{\Longrightarrow} y \in \mathit{FogSet}(z) \wedge z \in \mathit{FogSet}(x) \tag{3.11}$$
$$\overset{\mathit{Transitivity}}{\Longrightarrow} y \in \mathit{FogSet}(x) \tag{3.12}$$
$$\Longrightarrow \mathit{FogSet}(y) \subseteq \mathit{FogSet}(x) \tag{3.13}$$

$$\mathit{FogSet}(x) \subseteq \mathit{FogSet}(y) \tag{3.14}$$
$$\mathit{FogSet}(y) \subseteq \mathit{FogSet}(x) \tag{3.15}$$

With z being the transitive connection between the two Fog Sets.

$$\Longleftrightarrow \mathit{FogSet}(X) = \mathit{FogSet}(Y) \tag{3.16}$$

3.2.6 Service Constraint

In addition to the previously introduced concepts, most architectures are limited to specific services which are offered within them. These services are summarized in the *Service Set* as shown in Definition 8.

Definition 8: Service Set

$$ServiceSet := \{s \mid s \text{ is a Service }\}$$

To address this, the following three service sets — Provide, Consume, and Interest — create sets based on the Fog Components relation to the given service as defined by their names (Definition 9).

Definition 9: Service Sets

$Provide(s) := \{x \mid x\ offers\ and\ advertises\ s \in ServiceSet\}$
$Consume(s) := \{x \mid x\ requests\ s \in ServiceSet\}$
$Interest(s) := \{\ Provide(s)\ \cup\ Consume(s)\ \}$

with:
  $x \in FogComponentSet$
  $s \in ServiceSet$

Using these definitions, we define the following sets P, C, and I, which present selections on the given Fog Concept with respect to the provided service as shown in Definition 10.

Definition 10: Service Constraint

$P_s(f(x)) := \{f(x) \cap Provide(s)\}$
$C_s(f(x)) := \{f(x) \cap Consume(s)\}$
$I_s(f(x)) := \{f(x) \cap Interest(s)\}$

with:
  $x \in FogComponentSet$
  $f(x) \in \{\ FogVisibility(x),\ FogHorizon(x),$
    $FogReachability(x),\ FogSet(x)\}$
  $s \in ServiceSet$

As in most cases, the interest of a Fog Component for a given service will be used most often, Definition 11 presents an abbreviated form:

Definition 11: Interest Abbreviation

$\mathsf{FogVisibility}_s(x) := I_s(\mathsf{FogVisibility}(x))$
$\mathsf{FogHorizon}_s(x) := I_s(\mathsf{FogHorizon}(x))$
$\mathsf{FogReachability}_s(x) := I_s(\mathsf{FogReachability}(x))$
$\mathsf{FogSet}_s(x) := I_s(\mathsf{FogSet}(x))$

with:
$x \in \mathsf{FogComponentSet}$
$s \in \mathsf{ServiceSet}$

Finally, Definition 12 allows us to specify several services which a Fog Concept f depends on instead of just single services.

Definition 12: Multi-Service Constraint

$P_{s_1,s_2,\dots,s_n}(f(x)) := \{f(x) \cap (\mathsf{Provide}(s_1) \cup \mathsf{Provide}(s_2) \cup \dots \cup \mathsf{Provide}(s_n))\}$
$C_{s_1,s_2,\dots,s_n}(f(x)) := \{f(x) \cap (\mathsf{Consume}(s_1) \cup \mathsf{Consume}(s_2) \cup \dots \cup \mathsf{Consume}(s_n))\}$
$I_{s_1,s_2,\dots,s_n}(f(x)) := \{f(x) \cap (\mathsf{Interest}(s_1) \cup \mathsf{Interest}(s_2) \cup \dots \cup \mathsf{Interest}(s_n))\}$

Abbreviated Form:
$f_{s_1,s_2,\dots,s_n}(x) := \{f(x) \cap (\mathsf{Interest}(s_1) \cup \mathsf{Interest}(s_2) \cup \dots \cup \mathsf{Interest}(s_n))\}$

with:
$x \in \mathsf{FogComponentSet}$
$f(x) \in \{\ \mathsf{FogVisibility}(x),\ \mathsf{FogHorizon}(x),$
$\mathsf{FogReachability}(x),\ \mathsf{FogSet}(x)\}$
$s \in \mathsf{ServiceSet}$

These limited sets are subsets of the corresponding Fog concept. Therefore, in the case of Interest (I) a Fog Concept limited to all available services will result in the original definition of the Fog Concept itself, and thus does not display a constraint, as shown in Definition 13.

Definition 13: Multi-Service Constraint

$$I_{s_1,s_2,\ldots,s_n}(f(x)) = f(x)$$

with:

$f(x) \in \{ \mathit{FogVisibility}(x),\ \mathit{FogHorizon}(x),\ \mathit{FogReachability}(x),\ \mathit{FogSet}(x)\}$

$s \in \mathit{ServiceSet}$

$n = |\mathit{ServiceSet}|$

This allows us to model Fog Architectures with several different requested services, as we will discuss in the following example, but also address the issue that on different levels in Fog Architectures different services might be of interest.

Figure 3.14 presents an example Fog Architecture. It shows several Fog Components in different contexts namely the Edge (A_1-A_5), Fog (B_1-B_4), and Cloud (C) and depicts the Fog Horizons as connections between Fog Components. Additionally, the Fog Components in the Fog context provide the Services s_1 and s_2 and the Fog Component C provides s_3. Based on this example, the following Equation 6 shows the corresponding sets for the service constraints reduced to the Fog Horizons and Fog Sets, as no Fog Visibilities are shown.

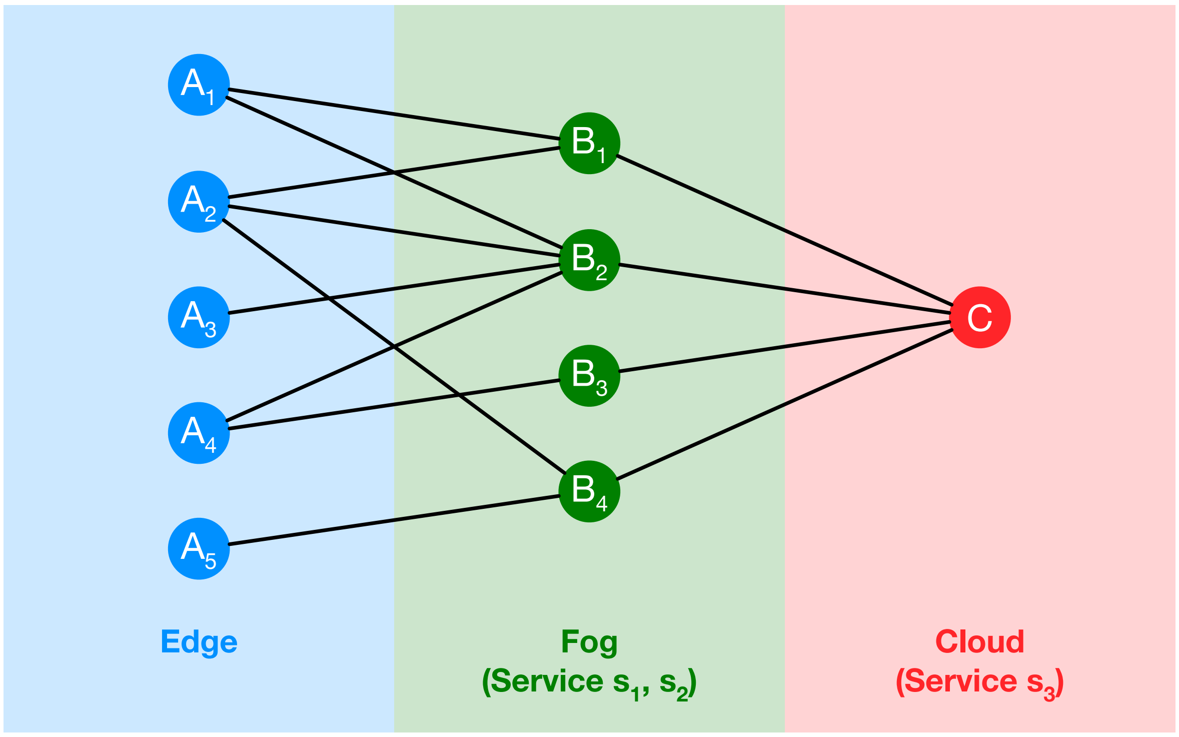


Figure 3.14: The figure shows an example setup for a 3-layered Fog Architecture with five edge nodes, four Fog Nodes and a single cloud node. Each Fog Node offers the two services s_1 and s_2 while the cloud offers service s_3. The connections between the Fog Components indicate their Fog Horizons.

Equation 6: Multi-Service Constraint Example

$$\mathrm{FogSet} = \{A_1, A_2, A_3, A_4, A_5, B_1, B_2, B_3, B_4, C\}$$

$$\begin{aligned}
&\mathrm{Provide}(s_1) = \{B_1, B_2, B_3, B_4\} = \mathrm{Provide}(s_2)\\
&\mathrm{Provide}(s_3) = \{C\}\\
&\mathrm{Consume}(s_1) = \{A_1, A_2, A_3, A_4, A_5\} = \mathrm{Consume}(s_2)\\
&\mathrm{Consume}(s_3) = \{B_1, B_2, B_3, B_4\}
\end{aligned}$$

$$\begin{aligned}
&\mathrm{FogHorizon}_{s_1}(A_1) = \mathrm{FogHorizon}_{s_2}(A_1) = \{B_1, B_2\}\\
&\mathrm{FogHorizon}_{s_1}(A_2) = \mathrm{FogHorizon}_{s_2}(A_2) = \{B_1, B_2, B_3\}\\
&\mathrm{FogHorizon}_{s_1}(A_3) = \mathrm{FogHorizon}_{s_2}(A_3) = \{B_2\}\\
&\mathrm{FogHorizon}_{s_1}(A_4) = \mathrm{FogHorizon}_{s_2}(A_4) = \{B_2, B_3\}\\
&\mathrm{FogHorizon}_{s_1}(A_5) = \mathrm{FogHorizon}_{s_2}(A_5) = \{B_4\}\\
&\mathrm{FogHorizon}_{s_1}(B_1) = \mathrm{FogHorizon}_{s_2}(B_1) = \{B_1\}\\
&\mathrm{FogHorizon}_{s_1}(B_2) = \mathrm{FogHorizon}_{s_2}(B_2) = \{B_2\}\\
&\mathrm{FogHorizon}_{s_1}(B_3) = \mathrm{FogHorizon}_{s_2}(B_3) = \{B_3\}\\
&\mathrm{FogHorizon}_{s_1}(B_4) = \mathrm{FogHorizon}_{s_2}(B_4) = \{B_4\}\\
&\mathrm{FogHorizon}_{s_3}(B_1) = \mathrm{FogHorizon}_{s_3}(B_2) = \mathrm{FogHorizon}_{s_3}(B_3) =\\
&\qquad\qquad\qquad\qquad \mathrm{FogHorizon}_{s_3}(B_4) = \{C\}
\end{aligned}$$

$$\begin{aligned}
&\mathrm{FogSet}_{s_1} = \{A_1, A_2, A_3, A_4, A_5, B_1, B_2, B_3, B_4\}\\
&\mathrm{FogSet}_{s_2} = \mathrm{FogSet}_{S_1}\\
&\mathrm{FogSet}_{s_1,s_2} = \mathrm{FogSet}_{s_1} = \mathrm{FogSet}_{s_2}\\
&\mathrm{FogSet}_{s_3} = \{B_1, B_2, B_3, B_4, C\}\\
&\mathrm{FogSet}_{s_1,s_2,s_3} = \mathrm{FogSet}
\end{aligned}$$

3.2.7 Set Relations

After introducing the different concepts, we compare the cardinalities of the respective sets and create an order. While all sets could potentially have an infinite cardinality, given a fixed amount of Fog Components, Figure 3.15 shows how the different sets relate to each other, excluding the sets *Provide* and *Consume* for clarity.

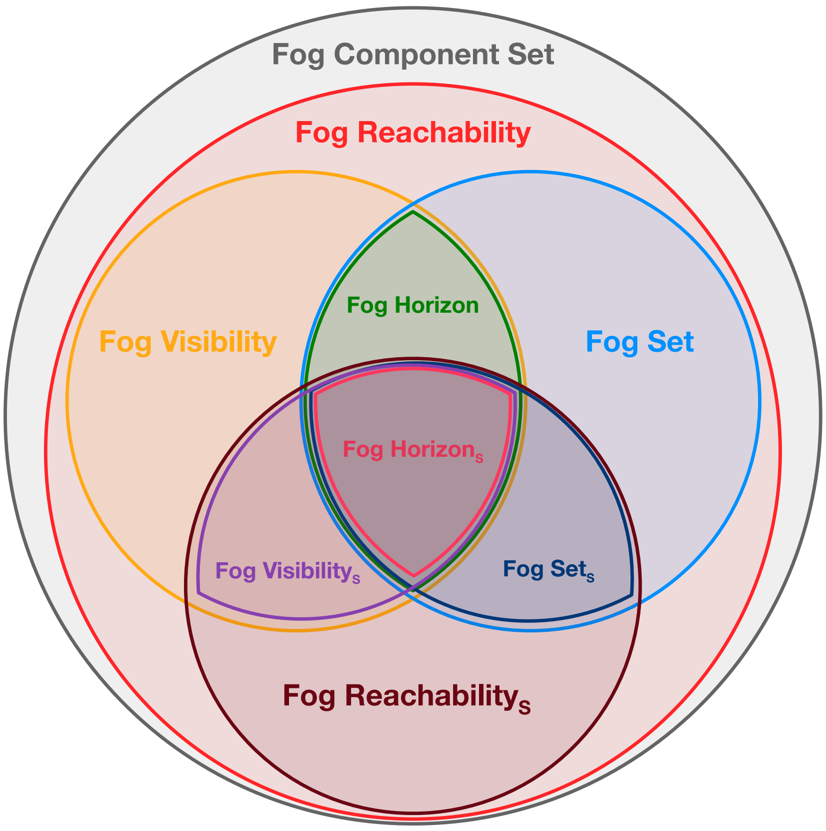


Figure 3.15: Relations between the different xFogCore set concepts adapted from Henze et al. [66]: The different sets are represented in different colored shapes showing their cardinality in comparison to the other sets. While we introduced the Provides, Consumes, and Interest sets and their relations to the different set concepts, we only present the generalized idea of limiting a set to one specific service in this diagram. Thus, including in addition to the basic sets, the service limited sets for the Fog Visibility, Fog Horizon, Fog Reachability, and Fog Set.

Looking at all available Fog Components in the `Fog Component Set`, the next biggest set is formed by the `Fog Reachability`. As already described in Section 3.2.4 it could in theory include the entire internet. Depending on the Fog Components, the next biggest set is either the `Fog Visibility` or the `Fog Set`. If X has many direct partners who could receive messages from it, the `Fog Visibility` could be the bigger sets. On the other hand, if many Fog Components are transitively connected and can equally exchange messages, the `Fog Set` might be the bigger set.

Therefore, a direct comparison of those two sets is not possible, but there is an overlapping area. This overlapping area is the `Fog Horizon`. It is a subset of the `Fog Visibility` and is, by definition, included in the `Fog Set`.

Finally, we want to address all the subsets created by limiting the concepts to a specific service S. Starting with the smallest set, the `Fog Horizon` limited by S is a subset of the `Fog Horizon` itself. The `Fog Visibility` limited by S includes the `Fog Horizon`$_s$ but additionally a part of the `Fog Visibility` which can be reached. The `Fog Set`$_s$ overlaps with the `Fog Visibility`$_s$ in the area of the `Fog Horizon`$_s$ but also includes the transitive connections for this set in the `Fog Set` area. The `Fog Reachability`$_s$ includes all three of those limited sets and additionally a part of the `Fog Reachability`. Equation 7 lists these relations of sets.

Equation 7: Set Relations

$$
\begin{aligned}
&FogComponentSet \supseteq FogReachability(x)\\
&FogReachability(x) \supseteq FogVisibility(x)\\
&FogReachability(x) \supseteq FogSet(x)\\
&FogVisibility(x) \cap FogSet(x) = FogHorizon(x)\\
&FogHorizon(x) \supseteq FogHorizon_S(x)\\
&FogVisibility(x) \supseteq FogVisibility_S(x)\\
&FogVisibility_S(x) \supseteq FogHorizon_S(x)\\
&FogSet(x) \supseteq FogSet(x)\\
&FogSet_s(x) \supseteq FogHorizon_s(x)\\
&FogHorizon_s(x) = FogVisibility_s(x) \cap FogSet_s(x)\\
&FogReachability(x) \supseteq FogReachability_s(x)\\
&FogReachability_s(x) \supseteq FogVisibility_s(x) \cap FogSet_s(x)
\end{aligned}
$$

3.3 Communication Set

The second set we want to investigate is the *Communication Set*, and therefore the interactions / connectors between the different components of the Fog Set. While most architectures use the same communication medium for the entire communication, Fog Architectures often involve different communication channels depending on the devices used. These channels can range from close proximity such as NFC or Bluetooth up to long distance communication such as 3G, 4G, or 5G. The following Figure 3.16 shows an excerpt of different communication channels which can be found within Fog Architectures including the channels used in the multi-case studies presented in the validation in Chapter 6.

In the diagram, the channels are placed within a 2x2 grid in which the columns indicate whether the channels are wired or wireless and the rows indicate local and remote proximity. As such, the placement within the grid described a double inheritance of the contained channel to the according superclasses indicate by row and column. Additionally, all superclasses are `Communication Channels` themselves. This set, called *Communication Channel Set*, is defined in Definition 14. It contains all potential communication channels that can be used within a Fog Architecture.

Definition 14: Communication Channel Set

$\mathit{CommunicationChannelSet} = \{c \mid c \text{ is a communication channel.}\}$

In the paper "Fog Horizons: A Theoretical Concept for Enable Dynamic Fog Architectures" [66] in the definition of Fog Visibilities, we mention that communication channels can be unidirectional or bidirectional. While most communication channels that are used in the context of Fog Architectures are bidirectional, network limitations as well as sensors only providing data can include unidirectional communication channels. To include and address these channels, our definition of Fog Visibility is unidirectional in comparison to the Fog Horizon and accordingly, the Communication Set can include unidirectional communication channels.

Unidirectional communication is closely linked to unidirectional networks. While unidirectional communication can include single connections between components to be only able to send data in one direction, unidirectional networks try to separate networks using software but also specialized hardware devices such as unidirectional network bridges. These bridges allow data flow in only one direction, and thus from one network to another one, which makes it physically impossible to transfer data back [137]. The separation of networks is mainly used as a security measurement, consisting of one highly secure network and an open counterpart which makes it very important for practical applications. Another application for

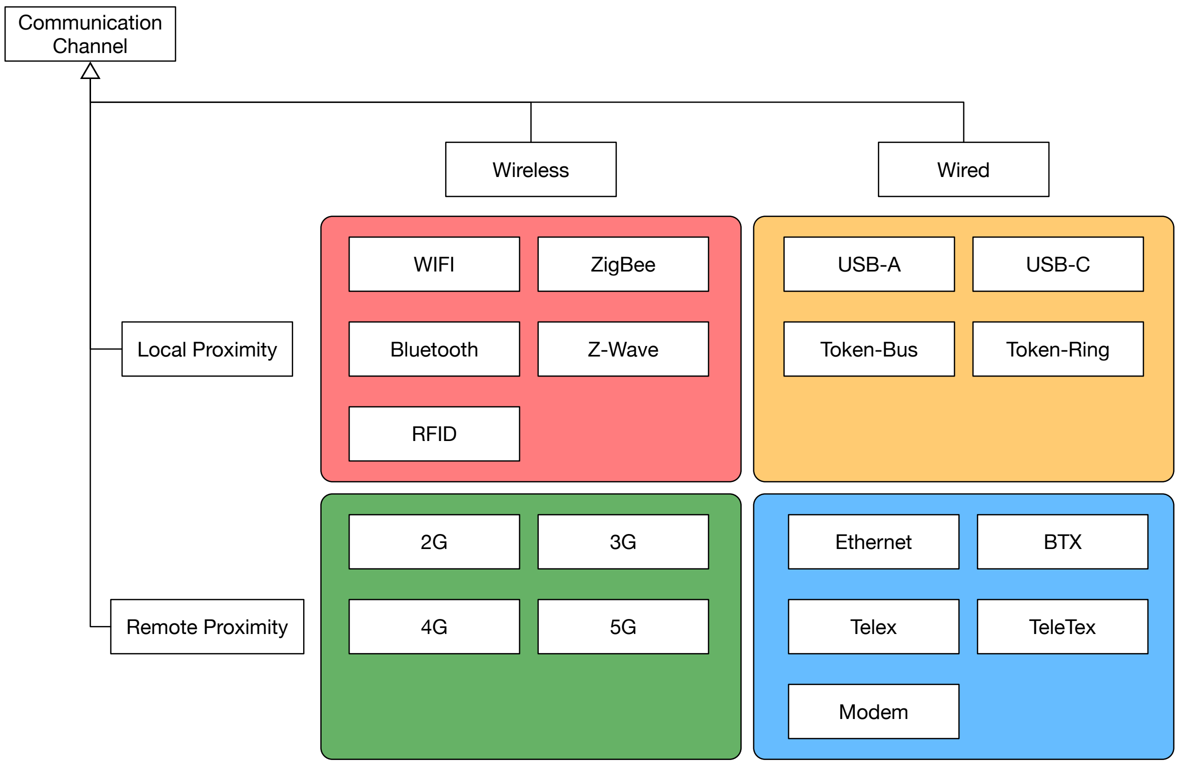


Figure 3.16: Excerpt of possible communication channels in the context of Fog Architectures: The channels are divided based on the physical medium that they are using (wired versus wireless), but also the physical distance for which the communication channels can be used.

unidirectional communication is for ”sensed information flow which plays a central role in schooling and flocking” [100].

Bidirectional communication on the other side describes most of the communication channels used in Fog Architectures. Also bidirectional communication channels can be easily limited to unidirectional communication using software solutions, channels such as Wifi, Bluetooth, or USB rely on an open bidirectional communication approach. Therefore, encapsulating a certain amount of trust to other participants within the communication structure is essential for Fog Architectures; only trusted instances are described as part of a Fog Architecture.

As shown in the metamodel in Figure 3.7, in addition to the equivalent of a *Component* in the context of a Software Architecture which is the Fog Component and the related sets, we define a *Communication Component* as the equivalent to the *Connector*.

A *Communication Component* is a triple that consists of the source Fog Component, the used *Communication Channel*, and the destination Fog Component as shown in Definition 15. Thus, the source and destination allow to define unidirectional communications.

Definition 15: Communication Component

$$\begin{aligned}\mathrm{CommunicationComponent} := \{&\mathrm{SourceFogComponent},\\ &\mathrm{CommunicationChannel},\\ &\mathrm{DestinationFogComponent}\}\end{aligned}$$

All *Communication Components* of one Fog Architecture are grouped within the *Communcation Set*. The *Communication Set* is defined as shown in Definition 16 and is the equivalent to the Fog Set. For a *Communication Component* to be considered part of the *Communication Set*, the *source* and *destination* Fog Component have to be part of the Fog Set, and the Communication *channel* has to be part of the *Communication Channel Set*.

Definition 16: Communication Set

$$\begin{aligned}\mathrm{CommunicationSet} := \{&c : \mathrm{CommunicationComponent} =\\ &(\mathrm{source}, \mathrm{channel}, \mathrm{destination}) \mid \mathrm{source} \in \mathrm{FogSet}\\ &\wedge \mathrm{destination} \in \mathrm{FogSet}\\ &\wedge \mathrm{channel} \in \mathrm{CommunicationChannelSet}\}\end{aligned}$$

3.4 Graph Representation

In addition to modeling a Fog Architecture as a software architecture, the combination of the two sets – Fog Set and Communication Set – enables the representation of a Fog Architecture as a graph which is one view on the components and connectors structure as introduced by Bass et al. [13].

3.4.1 Graph Definition

Representing Fog Architectures as graphs provides the major advantage of getting an overview of what the architecture looks like, what elements they consist of, and in which way they are connected. While all the information is already existing within the sets, the visual representation makes the sets easier accessible. In such a graph, all elements of the Fog Set are represented as the nodes of the graph while the elements of the Communication Set represent the edges.

Figure 3.17 shows a generalized representation of a Fog Architecture that allows a unified syntax for the nodes, edges, and paths throughout the graph. While this representation assumes that all nodes on higher layers are reachable by all elements of lower layers, this is often not the case in Fog Architectures. In those architectures, only a subset of nodes can reach a subset of nodes on the higher layer and so on. Those subsets of nodes would create true subgraphs of the current graph representation. As all subgraphs display the same layout as the current graph, we could then use the current graph to describe those subgraphs. Therefore, the generalized representation can be used to introduce the following concepts.

In the graph, the colors and shapes represent the concept of a Fog Architecture having three different subsets; *blue*, *squared* nodes represent the set of all Edge Devices, *red*, *circled* nodes being Fog Nodes, and *black*, *diamond-shaped* nodes showing Cloud Devices. The *grey* boxes gather nodes of the same layer into sets that are labeled as X for Edge Devices, A_i for Fog Node layers, and Y for the Cloud Layer. While the nodes within X and Y are continuously labeled, the first index of the nodes in the different Fog Layers indicates the node number and the second index indicating the layer number. For simplicity, only the first few nodes on each layer and only the first few layers are shown; the indices *k, l , m,* and *n* and their indices reflect the cardinality of each parameter, and therefore the amount of devices and layers. Additionally, there are two types of connections: Solid lines represent direct connections between the connected nodes, dotted lines on the other hand indicate that those nodes are connected as well, but many nodes or layers might not be displayed in between. As an example, the nodes X_2 and X_3 are not shown, same as for the layers A_3 and A_4; those layers are only implicitly given. In order to make the naming consistent, the Fog Layer's index starts at 1, while the nodes indices start at 0, to address the idea that the Edge Layer is layer 0.

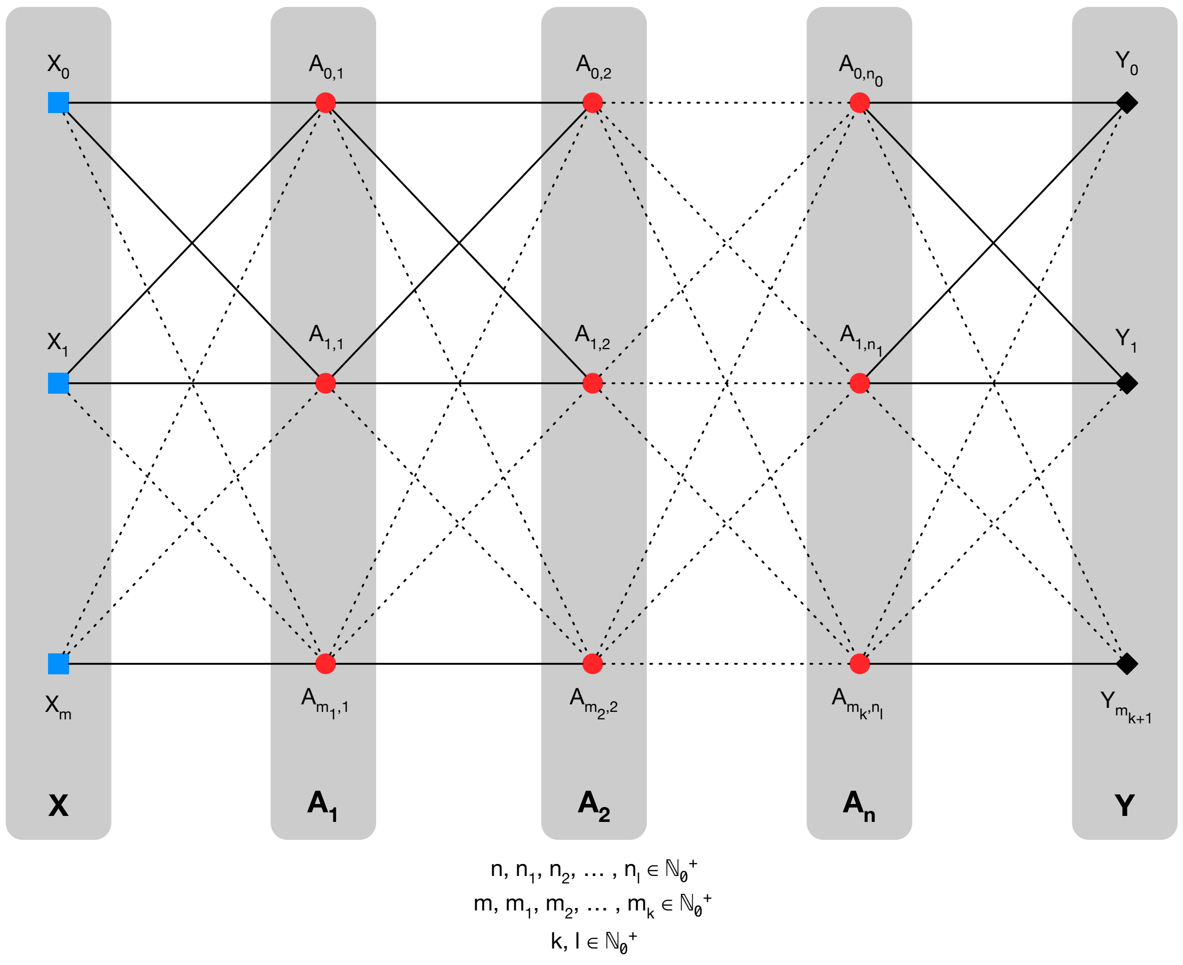


Figure 3.17: Generalized model of a Fog Architecture as a graph representation. The colors and shapes of the Fog Components represent different layers of the Fog Architecture. The *blue, square* nodes which are labeled $X_1 - X_m$ are the Edge Devices of the architecture, the *circle* nodes $A_{1,1} - A_{m_k,n_l}$ in *red* represent Fog Nodes, and the *black, diamond-shaped* nodes labeled $Y_1 - Y_{m_{k+1}}$ represent Cloud Devices.

On the other hand, the Cloud Layer can be labeled as the $n+1$ layer with n being the maximum amount of Fog Layers. We will use this idea later on to simplify the notation of some formulas, while we use X and Y in this context to clearly separate the different scopes of the layers.

Definition 17 shows the mathematical description of Figure 3.17: defining the Fog Architecture as a set of nodes and edges, as well as showing the resulting node and edge sets. The representation of the node set shows the listing of the nodes in a specific way: the columns of the set are horizontal paths through the graph, while the rows represent the layers of the graph. This allows us to define the node set as a union of the sets of all layers. The edges can be described as the union of all cartesian products of neighboring layers creating a partitioning of the graph.

Definition 17: Graph Definition – Fog Architecture

Fog Architecture $:= G = \{V, E\} = \{FogSet, CommunicationSet\}$

$$V = \left\{ \begin{array}{ccccc} X_0, & X_1, & X_2, & \dots, & X_m, \\ A_{0,1}, & A_{1,1}, & A_{2,1} & \dots, & A_{m_1,1}, \\ A_{0,2}, & A_{1,2}, & A_{2,2} & \dots, & A_{m_2,2}, \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{0,n_0}, & A_{1,n_1}, & A_{2,n_2} & \dots, & A_{m_k,n_l}, \\ Y_0, & Y_1, & Y_2, & \dots, & Y_{m_{k+1}} \end{array} \right\} = X \cup A_0 \cup A_1 \cup \dots \cup A_n \cup Y$$

$$E = (X \times A_0) \cup (A_0 \times A_1) \cup \dots \cup (A_n \times Y)$$

This partitioning also leads to another representation of the Fog Architecture graph as shown in Figure 3.18. In this figure, the graph is separated into subgraphs connecting neighboring layers which allows us to define a Fog Architecture as a set of bipartite graphs. Based on our assumption that every node of a higher layer is reachable by every node on the lower level, those subgraphs are in fact complete bipartite graphs. The connections between neighboring layers, seen as a relation, are surjective or right total meaning that every element of the higher layer is reachable by at least one node in the lower layer and left total or definal meaning that every element of the lower level can be reached by at least one node of the higher level, making the relations bi-total [12].

The idea of representing the Communication Set as a set of relations between neighboring layers, which are defined within the cartesian product of those, also allows us to pick up the idea of service constraints as introduced in Section 3.2.6 again.

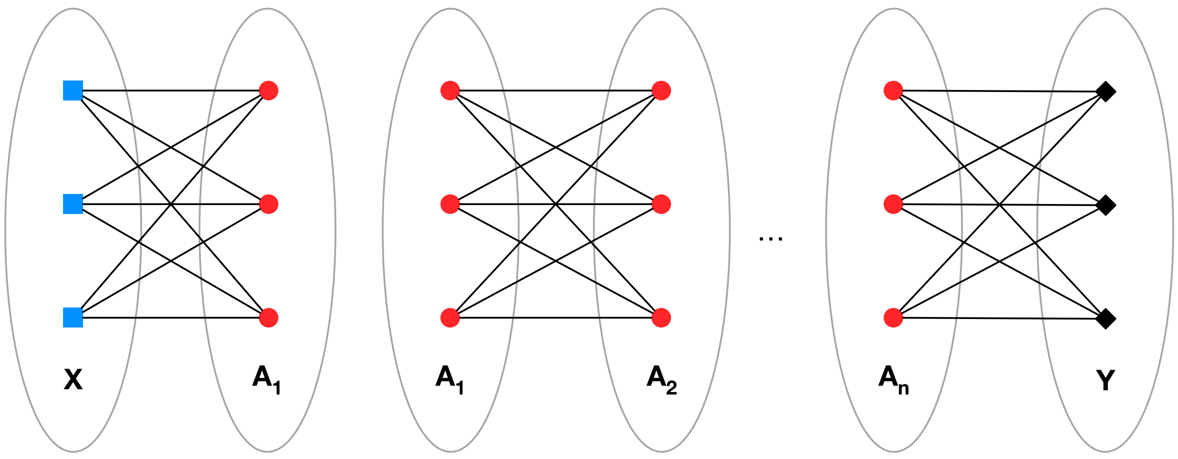


Figure 3.18: The figure presents a set of bipartite graphs of a Fog Architecture with Edge Devices on the leftmost layer, three layers of Fog Nodes and one layer of Cloud Devices on the right. In accordance to Figure 3.17, Edge Devices are *blue squares*, Fog Nodes *red circles*, and Cloud Devices *black diamonds-shaped* nodes.

As shown in Definition 18, we can create a relation as a subset of the cartesian product of two neighboring layers that includes all edges that start in the lower layer and end in the higher layer when the Fog Component of the higher layer is part of the Provides set. Therefore, we define a set of relations consisting of one relation per neighboring layer pair that creates a subgraph of the Fog Architecture limited to service s. We rely on this idea and the concept of bipartite graphs in Chapter 4.

Definition 18: Service Constraint – Relation

$R_s \subseteq A_n \times A_{n+1} := \{(a, b) \mid a \in A_n \ \wedge \ b \in A_{n+1} \ \wedge \ b \in \mathsf{Provides}(s)\}$

with:

$a, b \in \mathsf{FogComponentSet}$
$s \in ServiceSet$

The syntax introduced by Figure 3.17 and Definition 17 also allows us to define a Fog Paths, a finite sequence of nodes. As shown in Definition 19, this path is a set of visited nodes, starting at one node within the Fog Architecture $A_{i,j}$ going up or down the layers until the final node $A_{l,j+l}$. The index l is part of $\mathbb{Z}^*$ which includes all integer values without 0; that ensures a path length of at least two nodes or one edge. While the node number within one layer can be different on each step, the layer count has to strictly increase or decrease by one each step. Fog Paths with increasing layer indices are directed paths towards the cloud, Fog Paths with decreasing layer indices are directed paths towards the edge. The syntax $\mathsf{FogPath}^{\leftrightarrow}$ followed by the set of Fog Components allows to define undirected paths.

Definition 19: Fog Path

$\mathsf{FogPath} := \{A_{i,j},\ \ldots,\ A_{k,j+l}\}$

with:

$i, j, k \in \mathbb{N}_0$

$l \in \mathbb{Z}^*$

3.4.2 Service Discovery & Service Offer View

Graph representations of software architectures are usually shown as undirected graphs. If edges have a direction, they are used to indicate in which direction data is sent. While this is one important aspect, another aspect is often not modeled: How are software architectures set up. The architecture setup can be accomplished in two ways: Service Discovery & Service Offering. *Service Discovery* describes, as defined in Definition 20, the approach how services can be discovered within a Fog Architecture which happens in a bottom-up approach indicating that lower level components search their Fog Horizon for a service of a higher level component. This would lead to a directed graph from lower level components to higher level components.

Definition 20: Service Discovery

Service Discovery is a directed graph in which lower level Fog Components can discover services of higher level Fog Components.

On the other side, *Service Offering* describe, as shown in Definition 21, the top-down approach of higher level Fog Components promoting their services within their Fog Horizon, making discoveries obsolete. Therefore, these graphs are directed from higher level Fog Components to lower level components.

Definition 21: Service Offering

Service Offer is a directed graph in which higher level Fog Components offer services to lower level Fog Components.

Inside a Fog Architecture, the use of different communication channels can also lead to different communication approaches. Especially for components that try to join the Fog Architecture, the information how to find services is of high importance and can already be presented by the Fog Architecture. Therefore, the Service Discovery versus Service Offering view of the system is particularly important in Fog Architectures and should be addressed as one view of the component and connector structure as presented by Bass et al. in [13].

3.4.3 Adjacency Matrix

The following Table 3.2 presents the adjacency matrix for the generalized Fog Architecture graph solely using Service Discovery or Service Offering as introduced in the previous section. The resulting matrix is sparse and symmetrically filled with values while the signs change. This matrix only reflects whether connections exist or not, it is not weighted, which results in a 1 if a connection exists from one node to another and a -1 if the reverse is true. All values highlighted in green indicate values that are positive for Service Discovery and negative for Service Offering. On the other side, red cells are negative for Service Discovery and positive for Service Offering. This results in the pattern as shown in the table in which all nodes of one layer are connected to all nodes of the lower layer and all of the higher layer connected to all on the lower layer just with different signs along the diagonal. All other cells to the left and right of those boxes are filled with zeros. Based on the communication channel used and the technology chosen, depending on the layers, some of the neighboring nodes might use Service Discovery, others Service Offering, but all will follow the given pattern.

Table 3.2: Adjacency Matrix for a generic Fog Architecture as shown in Section 3.4. In case of Service Discovery, the cells colored green have positive signs and the cells colored red have negative signs (Bottom-Up). In the case of a Service Offer, the green cells have negative signs and the red cells have positive signs (Top-Down).

	X_1	X_2	...	X_m	$A_{1,1}$	$A_{2,1}$	...	$A_{m_1,1}$	$A_{1,2}$	$A_{2,2}$	...	$A_{m_2,2}$
X_1	0	0	0	0	±1	±1	±1	±1	0	0	0	0
X_2	0	0	0	0	±1	±1	±1	±1	0	0	0	0
...	0	0	0	0	±1	±1	±1	±1	0	0	0	0
X_m	0	0	0	0	±1	±1	±1	±1	0	0	0	0
$A_{1,1}$	∓1	∓1	∓1	∓1	0	0	0	0	±1	±1	±1	±1
$A_{2,1}$	∓1	∓1	∓1	∓1	0	0	0	0	±1	±1	±1	±1
...	∓1	∓1	∓1	∓1	0	0	0	0	±1	±1	±1	±1
$A_{m_1,1}$	∓1	∓1	∓1	∓1	0	0	0	0	±1	±1	±1	±1
$A_{1,2}$	0	0	0	0	∓1	∓1	∓1	∓1	0	0	0	0
$A_{2,2}$	0	0	0	0	∓1	∓1	∓1	∓1	0	0	0	0
...	0	0	0	0	∓1	∓1	∓1	∓1	0	0	0	0
$A_{m_2,2}$	0	0	0	0	∓1	∓1	∓1	∓1	0	0	0	0

Chapter 4

xFogPlus: Dynamic and scalable Fog Computing

"The state of a fog network is dynamically changing due to the on-off switching of IoT applications and the mobility of fog nodes, as well as the unreliable access links of some fog nodes to the network. Fog computing should be autonomous to tackle the dynamics."

— YUXUAN JIANG [74]

xFogCore, as defined in Chapter 3, enables dynamics and scalability, but does not inherently support them. This is addressed by the second part of xFog: xFogPlus.

The first aspect of a Fog Architecture that can be dynamic are the Fog Components themselves. For instance, using the infrastructure example shown in the introduction in Chapter 1, traffic participants, such as cars, can join and leave a Fog Architecture. It results in dynamic behavior for the set of Fog Components that are considered part of the Fog Architecture, and therefore all core concepts of xFog introduced in Chapter 3 with xFogCore. This first part is addressed in Section 4.1 and discusses adding new components to the Fog Architecture as well as adding Fog Components to the layers of Fog Architectures. Therefore, we fulfill the functional requirements FR2 and FR4 relating to the *Technical Research Goal 2*.

Section 4.2 focuses on the second dynamic aspect of Fog Architectures that also enables scalability: layers. On the one side, new services can be added to existing layers, but also entire new layers can be established, which is requested by FR5 and NFR2 relating to the *Technical Research Goal 3*. Using new definitions for the different layers of a Fog Architecture, we address how layers can be added. Additionally, we introduce the idea of different *Views* for stakeholders within Fog Architectures to handle complexity and promote the highlighting of different aspects of the architecture which are of current interest.

4.1 Dynamic Fog Components

Dynamically adding and removing components from a Fog Architecture has two different aspects to it. On the one side, the concept needs to support the approach from a theoretical point of view by building on concepts that are inherently dynamic and provide methods to add and remove components. On the other side, from a practical point of view, the selected devices and selected techniques for discovery and integration need to be able to support the proposed theoretical concepts.

We already shortly addressed the communication channels in Section 3.3, which are the technical basis for the discovery and integration. On top of them, different discovery mechanisms can be deployed to enable finding devices that were added to the Fog Architecture or removing those that left it. We select a discovery mechanism that supports the theoretical ideas without putting an emphasis on the selection process. The same holds true for the communication channels; they are shown for the sake of completeness, but need further investigation.

The devices on the other hand also need to be flexible in the mechanisms used to connect to communication partners, and therefore mainly focus on wireless connectivity and open protocols. Wired components that are directly connected to a single component on the higher layer are less suited for dynamic Fog Component interaction, as the amount of communication partners is inherently limited. The same is true for devices that only use one single protocol; in general they are less likely to work in dynamic settings. As sensors, actuators, and all the devices on the way to the Cloud get more and more powerful, and therefore general-purpose devices as stated by Xu et al., Brody et al., and Shi et al. [20, 135, 153], we base our solution on the assumption that the devices are general-purpose devices free of protocol limitations and able to perform own calculations.

In order to add or remove components to or from a Fog Architecture, we have to address two separate issues. First, we have to discuss general ways of adding a component to a Fog Architecture and second, how components are added on specific layers.

4.1.1 General Addition of Components

Being able to add new components to a Fog Architecture is simplified based on the overarching mathematical definitions introduced in Chapter 3. Based on those definitions, especially Definition 2 and Definition 6, each component that should be considered part of the Fog Architecture needs to fulfill two requirements. First, the component needs to be part of the Fog Component Set and second, it needs to be part of the Fog Set: As all concepts introduced in Chapter 3 are based on the Fog Component Set, it is a mathematical necessity for all of them, including the Fog Set, but the Fog Set is only sufficient for the Fog Component Set.

The Fog Component Set itself already poses a problem. Based on the definition shown in Section 3.2.1, the Fog Component Set consists of all potential Fog Components. While Fog Components can be any IoT device on MOF level M0 and M1, the Fog Component itself is an instance of the Fog Type on M3, which in turn has the subclasses Edge Device, Fog Node, and Cloud Device. Although this definition is helpful if different Fog Architectures are available and they should be differentiated between each other, it is the wrong way around if new components should be added. In order to be considered part of the Fog Component Set, the component needs to be a Fog Component, and therefore already part of a Fog Architecture which is not the case for new components.

Therefore, to be able to add components to the Fog Component Set, an alternative definition for a Fog Component is required which solely focuses on properties of the component itself. The first hard requirement is that every Fog Component is necessarily an IoT device. This means, according to the definition provided in Section 2.1.1, that a component needs to have the capability:

1. to be interconnected,
2. to have the intention to share information across platforms,
3. to be uniquely addressable, and
4. to have computational capabilities.

Soft requirements are that the components:

1. preferably use wireless communication,
2. have an interest in locality,
3. have general-purpose computational power, and
4. which they offer as services to other components.

This definition allows us to extend the Fog Component Set by new components which are not involved in any Fog Architecture, yet.

The second requirement is that the component satisfies the definition of a Fog Set, and thus can be included in a Fog Architecture. Based on Definition 6, for a *Fog Component* x to be included in a *Fog Set*, the *Fog Component* needs to be in the transitive closure of the *Fog Horizon* of a *Fog Component* within the Fog Architecture that the *Fog Component* should be added to. Accordingly, it is mathematically sufficient for the *Fog Component* to be able to send and receive direct messages to and from any single *Fog Component* in the Fog Architecture.

The only other dependency the Fog Set indirectly depends on, via the Fog Horizon, and thus via the Fog Visibility, is the communication channel which is used to transmit the required messages for the Fog Visibilities. As we do not have any hard requirements for the communication channel selection, we do not have to further adjust our definition.

While this alternative definition of a Fog Component allows the addition of new components to a given Fog Architecture, from a Fog Computing and architectural design perspective it is unclear *where*, respectively, on which layer, the new Fog Component is added. This is addressed in the following Section.

4.1.2 Specific Addition of Components to Layers

After adding new components to the Fog Component Set and Fog Set, we have to address the issue on which layer or tier the component is added. While layers refer to the abstract concept, tiers are instances of those layers which we use when a Fog Architecture is instantiated to map layers to hardware components. As the core concept of Fog Computing is the deployment of the different layers closer and closer to the Edge, every layer is deployed on separate hardware components, introducing a one to one mapping: Every layer is deployed on one tier and every tier refers to one single layer. This allows us to use the terms layer and tier interchangeably. Based on the conceptual nature of xFog, and therefore xFogCore, xFogPlus, and xFogStar, we decided to use the conceptual term *layer* in our definitions instead of the term *tier*, which relates to concrete instances. Nevertheless, in the validation in Chapter 6, we address the difference between layers and tiers.

In the view of Fog Computing, components can be assigned to three layers: the Edge Layer, the Fog Layer, or the Cloud Layer. While the Edge Layer and Cloud Layer consist of single layers, the Fog Layer can consist of several individual layers.

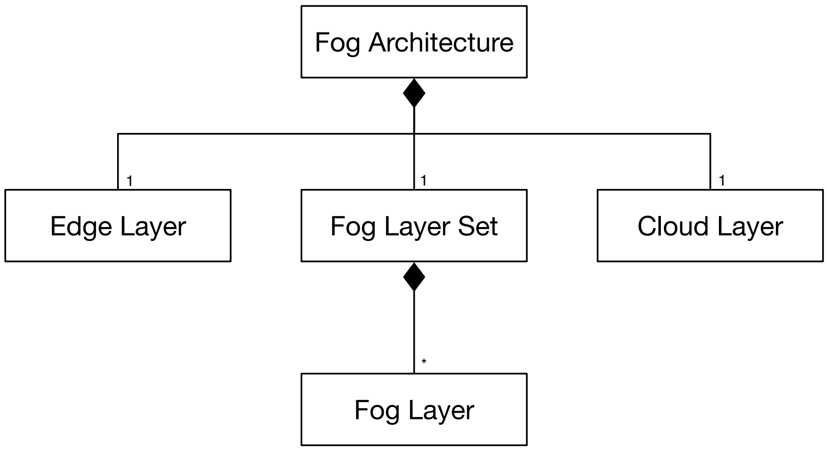


Figure 4.1: This diagram shows another viewpoint on the *Fog Architecture* compared to the definition shown in Chapter 3. It focuses on a higher abstraction level which is based on layers instead of individual components. The *Fog Architecture* consists of one *Edge Layer*, one *Fog Layer Set*, which in turn consists of several *Fog Layers*, and one *Cloud Layer*.

The representation shown in Figure 4.1 approaches the definition of Fog Architectures based on layers instead of components and connectors.

To indicate that the Fog Layer can consist of several layers, we rename the Fog Layer to *Fog Layer Set* in compliance with the introduced concept of xFogCore. The Fog Layer Set can consist of several Fog Layers itself.

We formalize Figure 4.1 as shown in Definition 22 and create a set which includes the layers of a Fog Architecture.

Definition 22: Layer-based Fog Architecture

$\mathsf{FogArchitecture} := \{\mathsf{EdgeLayer}, \mathsf{FogLayerSet}, \mathsf{CloudLayer}\}$
$|\mathsf{FogArchitecture}| = 2 + |\mathsf{FogLayerSet}|$

Second, we formalize the definition of the Fog Layer Set as shown in Definition 23. The Fog Layer Set depends on the current Fog Architecture, it should be evaluated for.

Definition 23: Fog Layer Set

$\mathsf{FogLayerSet} := \{\mathsf{l} \mid \mathsf{l} \text{ is a Fog Layer}\}$

As we only used the idea of layers in the graph concept in Section 3.4 so far, in the following, we want to present set definitions for the concept of layers by looking at their Fog Components. First, we present definitions for the different types of layers, followed by a generalization of the layer idea.

Definition 24 shows the properties of a *Fog Component* to be considered part of the *Edge Layer*. Each *Fog Component* x needs to be part of the *Fog Set*, thus, part of the *Fog Architecture*, and does not provide any services to other *Fog Components*, which is represented by not having any service s which makes *Fog Component* x part of its provide set.

Definition 24: Edge Layer

$$\mathsf{EdgeLayer} := \{x \mid x \in \mathsf{FogSet} \ \land \ \nexists s \in \mathsf{ServiceSet} : x \in \mathsf{Provide}(s)\}$$

The *Fog Layer* is defined as every *Fog Component* x that is, equal to the *Edge Layer*, part of the *Fog Set* and for which at least two services s_1 and s_2 exist so that *Fog Component* x consumes one of the services and offers the other one, as shown in Definition 25. This describes the idea that *Fog Components* in the *Fog Layer* bring services of higher layers, e. g., the *Cloud Layer*, closer to the *Edge Layer* but also do their own calculations.

Definition 25: Fog Layer

$$\begin{aligned}\mathsf{FogLayer} := \{x \mid x \in \mathsf{FogSet} \ \land \ \exists s_1,\ s_2 \in \mathsf{ServiceSet} :\\ x \in \mathsf{Consume}(s_1) \ \land \ x \in \mathsf{Provide}(s_2)\}\end{aligned}$$

The *Cloud Layer*, as shown in Definition 26, includes every *Fog Component* that does not consume any service itself.

Definition 26: Cloud Layer

$$\mathsf{CloudLayer} := \{x \mid x \in \mathsf{FogSet} \ \land \ \nexists s \in \mathsf{ServiceSet} : x \in \mathsf{Consume}(s)\}$$

Combining these three layers provides us with a definition for the idea of a layer as shown in Definition 27. The first definition is based on the idea that a layer is either an *Edge Layer*, a *Fog Layer*, or a *Cloud Layer*; while the second definition takes a closer look at these layers and extracts the common properties. Therefore, a *Layer*, independent of its type, includes all *Fog Components* that either have a common consumed service s or a common provided service s. It is important for layers to look either at Fog Components that provide the same service **or** consume the same service, but never both at the same time. Otherwise service providers and service consumers would be added to the same layer.

Definition 27: Layer

$$\begin{aligned}\mathsf{Layer} &:= \mathsf{EdgeLayer} \ \lor \ \mathsf{FogLayer} \ \lor \ \mathsf{CloudLayer}\\ &= \{x \mid x \in \mathsf{FogSet} \ \land \ (\exists s \in \mathsf{ServiceSet} : x \in \mathsf{Provide}(s) \ \lor\\ &\qquad \exists s \in \mathsf{ServiceSet} : x \in \mathsf{Consume}(s))\}\end{aligned}$$

As already addressed, the Fog Layer Set consists of several layers. In Definition 28, we provide a possibility to distinguish between the layers within the Fog Layer Set based on the involved services. Each *Fog Layer* in the Fog Layer Set is defined by a service pair s_i, s_j that is on the one side consumed by the layer and on the other side provided by the layer. If different layers include the same Fog Components although being defined by different service pairs, those layers are fused to one layer consuming or providing several services.

Definition 28: Service specified Fog Layer

$$\begin{aligned} \mathsf{FogLayer}(s_i, s_j) := \{x \mid x \in \mathsf{FogLayer} \wedge i, j \in \{1, \dots, |\mathsf{ServiceSet}|\} \wedge \\ x \in \mathsf{Consume}(s_i) \wedge x \in \mathsf{Provide}(s_j)\} \end{aligned}$$

Using the provided definitions for the different layers, the issue of adding Fog Components to specific layers can be reduced to these Fog Components providing or consuming the services that uniquely identify each layer.

We represent the mapping between the provided and consumed services to the different layers using the *Service Set Mapping*. The Service Set Mapping consists of a set of triples. Each triple contains the set of consumed services as the first element, the layer which is described by the services as the second element, and the provided services as the third element:

$$(\{\mathsf{ConsumedServices}\}, \mathsf{Layer}, \{\mathsf{ProvidedServices}\})$$

As those triples can stretch across multiple lines, for readability, we highlight the different elements using colors: The consumed services are highlighted in red, the layer itself in green, and the provided services in blue. The order of the elements describes the typical Fog Computing layout with the Cloud Layer on top of the architecture representation and the Edge Layer at the bottom: Accordingly, the consumed services, which are provided by the layer above are left of the layer and the provided services are on the right.

In order to define the Service Set Mapping, we have to introduce a relation that takes an integer as a parameter and returns the according layer as shown in Definition 29. The Edge Layer represents the 0-th layer while the Cloud Layer is the m-th layer with m being the amount of layers in the Fog Architecture minus 1.

Definition 29: Layer Selection Relation

$$\mathrm{Layer}(n) := \begin{cases} \mathrm{EdgeLayer}, & \text{if } n = 0 \\ \mathrm{CloudLayer}, & \text{if } n = m \\ \mathrm{FogLayer}_n, & \text{if } n \neq 0, n \neq m \end{cases}$$

with:

$m = |\mathrm{FogArchitecture}| - 1$

$n \leqslant m$

Using this relation, we can define the Service Set Mapping as shown in Definition 30.

Definition 30: Service Set Mapping

$\mathrm{ServiceSetMapping} :=$

$$\bigcup_{i=m}^{0} (\{\mathrm{ConsumedServices}\}, \mathrm{Layer}(i), \{\mathrm{ProvidedServices}\})$$

with:

$m = |\mathrm{FogArchitecture}| - 1$

Figure 4.2 shows an example setup of different Fog Components distributed over the three layers: `Edge Layer`, `Fog Layer Set`, and `Cloud Layer`. The `Fog Layer Set` consists of three layers itself: `Fog Layer 1`, `Fog Layer 2`, and `Fog Layer 3`.

Although, in this example, all Fog Components in the `Edge Layer` consume the same service s_1, they could also consume other services as long as they do not offer any services themselves. The first two `Fog Layers` are examples for multiple consumed or provided services (s_2, s_3). These two layers present an instance of fused `Fog Layers` as the service pairs s_1, s_2 as well as s_1, s_3 result in the same set of Fog Components, and therefore are on the same layer. The third `Fog Layer` provides service s_4 and consumes s_5 which is provided by the `Cloud Layer`.

Additionally, the example shows the case that `Fog Component` F_7 (①) is added to the Fog Architecture. Based on the provided and consumed services s_2, s_3 and s_4, the new Fog Component is added to the second `Fog Layer`; although no connections are established, yet.

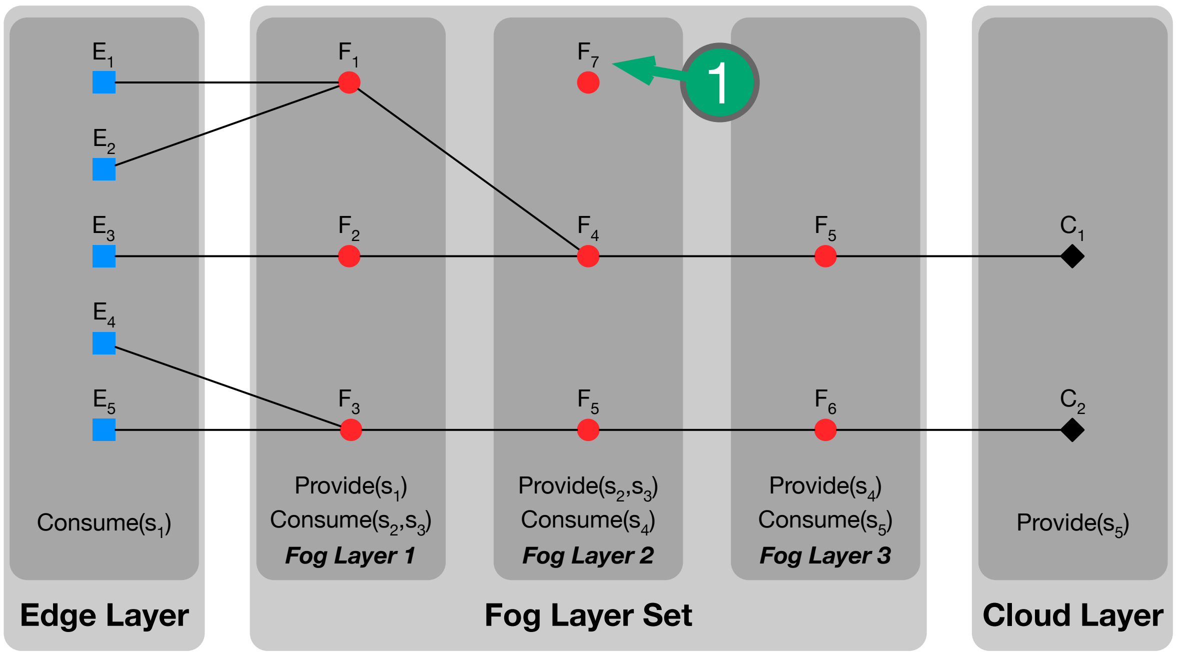


Figure 4.2: The graph shows an example for a Fog Architecture which is distributed over the three layers *Edge Layer*, *Fog Layer Set*, and *Cloud Layer*. While all *Fog Components* in the *Edge Layer* consume service s_1 this does not have to be the case; as long as they do not offer any services on their own, they are considered part of the *Edge Layer*. *Fog Layers* 1 and 2 provide examples for fused layers with two consumed and provided services, respectively. Although no connections are established, yet, the newly added *Fog Component* F_7 (①) can be assigned to *Fog Layer 2* based on the provided and consumed services s_2, s_3, and s_4.

4.2 Scalable Fog Architectures

After investigating individual Fog Components and how to add them to a Fog Architecture on any layer in the previous section, this section takes a closer look at the introduced layer definitions and their implications. First, we address how to setup a Fog Architecture by adding new layers and how to keep the Fog Architecture scalable by dynamically adding layers. We then present how different *Views* can affect Fog Architectures and how this can be exploited by refining the Fog Computing idea. Finally, we investigate the resulting implications on individual Fog Components.

4.2.1 Adding Layers

Adding new layers to existing architectures is one key aspect for making them dynamic and scalable. As most applications nowadays use client-server architectures, it is also essential for the transition from a centralized approach to a decentralized approach using Fog Computing. We differentiate between three different types of layer additions.

Starting from a client-server architecture, as shown in Figure 4.3, the first addition, highlighted by ①, is a layer in between the `Client Layer` and the `Server Layer` to bring the power of central instances closer to the consumer [17, 18]. In order to add this layer, at least one new Fog Component has to be added to the Fog Architecture, which provides the former service provided by the `Server Layer` and consumes an adjusted service from the `Server Layer` that only includes parts of the service which need to be executed in a centralized way. From the perspective of the *Client Layer*, nothing changes except that a new Fog Component offers the requested service with a potentially faster response time.

The second addition, as shown in Figure 4.4, focuses on adding a new layer to an existing Fog Architecture by introducing a new intermediate layer between the `Edge Layer` and the first `Fog Layer`, the last `Fog Layer` and the `Cloud Layer`, or in between two `Fog Layers`. All three of these approaches are similar in that sense, that they do not change the conceptual representation of the Fog Architecture. In each approach, a new `Fog Layer` is added; either as the new first `Fog Layer`, or as a new last `Fog Layer`, or as a `Fog Layer` in between. ① shows the position where the new layers are added for each subfigure. The required adjustments are the same as for the first approach: Providing the service that was previously provided by the higher layer and consuming a new, adjusted service that can only be evaluated on a more central *Fog Component*. Thus, the first addition is a special case of the second addition with the difference that the first addition includes a conceptual change from a 2-layered client server architecture to a 3-layered Fog Architecture.

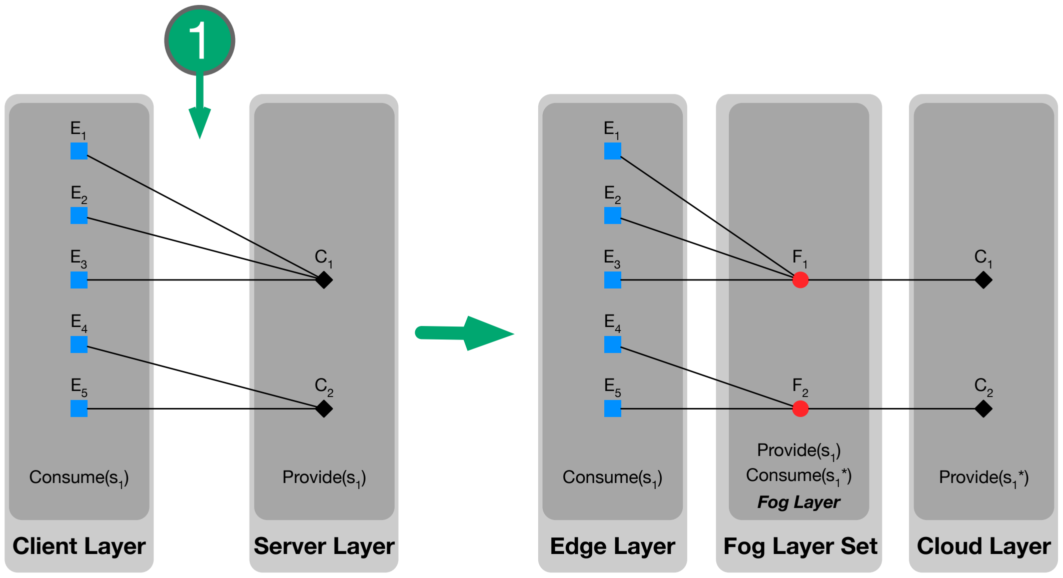
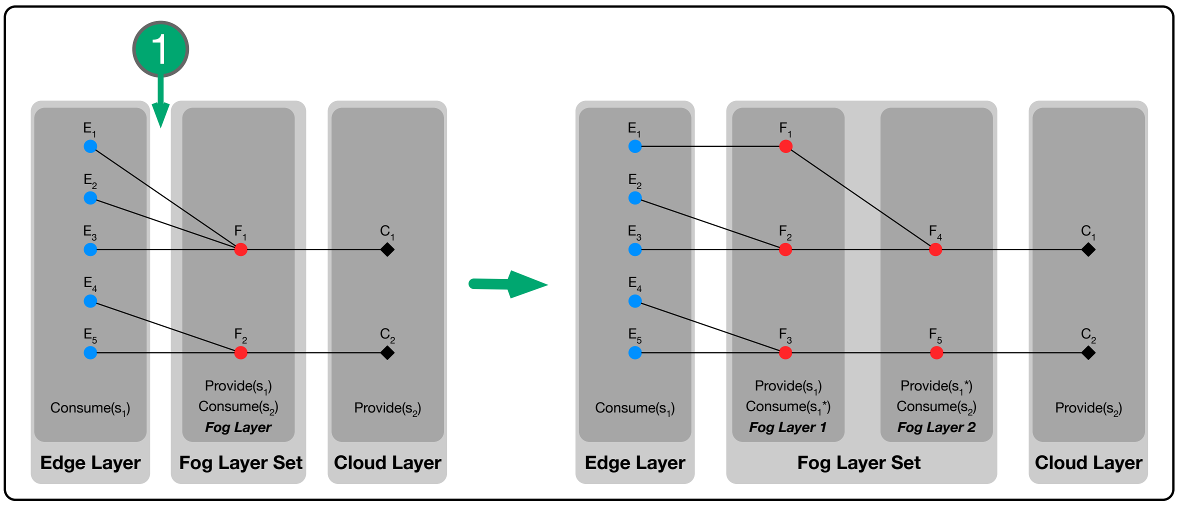
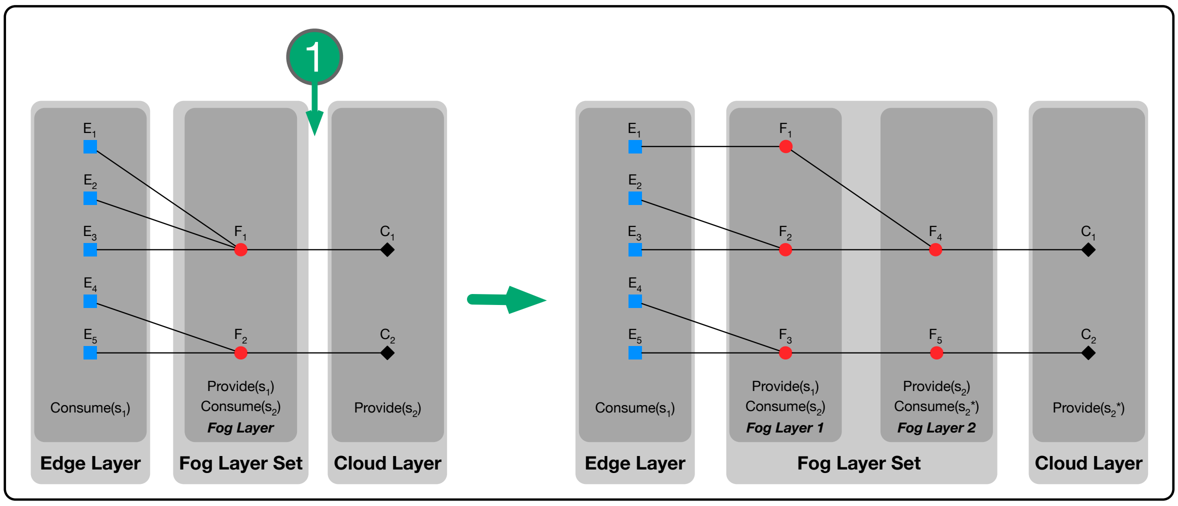


Figure 4.3: The example shows the first layer addition approach, which transitions a 2-layered client-server architecture into 3-layered Fog Architecture. For this addition, the new layer is inserted in between the *Client Layer* and the *Server Layer* (①). The exemplary resulting Fog Architecture with two new *Fog Components* F_1 and F_2 is shown on the right. The added *Fog Layer* provides service s_1 which was previously offered by the *Server Layer* and the *Server Layer* adjusts its service to s_1^* addressing parts of s_1 which need a higher level abstraction.

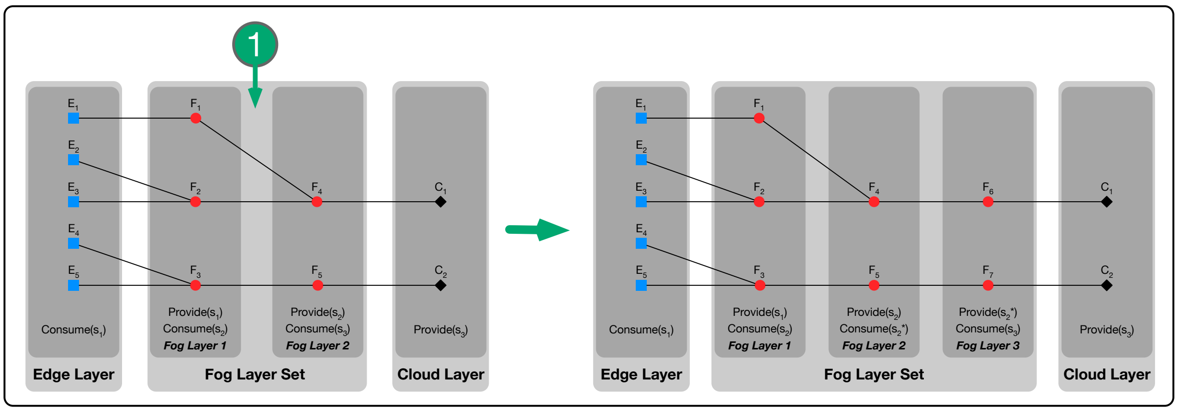
The third addition can be used for both, adding a *Fog Layer* to a client-server architecture or introducing new layers in an existing Fog Architecture as shown in Figure 4.5. It adds a layer by including Fog Components as new Edge Devices requesting services from the former `Edge Layer` (①) or by using additional Fog Components as a new `Cloud Layer` (②). Implementing this change only requires to add a new service to the `Edge Layer` which is consumed by the new layer or by providing a new service which is used by the former `Cloud Layer`. While it appears to be the easiest addition, as only one service has to be added and consumed, from a conceptual perspective, it has the biggest implications: A previous `Edge Layer` or `Cloud Layer` is converted into one layer of the `Fog Layer Set`.



(a) Adds the new *Fog Layer* in between the *Edge Layer* and the initial *Fog Layer* (①).



(b) Adds the new *Fog Layer* in between the last *Fog Layer* and the *Cloud Layer* (①).



(c) Adds the new *Fog Layer* in between two *Fog Layers* (①).

Figure 4.4: The example graph presents the three types of layer additions for the second approach which adds new layers to the *Fog Layer Set*. In each type, the left graph shows the previous architecture and the right graph the changed Fog Architecture. The services marked with a * indicate services that were changed due to the introduction of an additional layer.

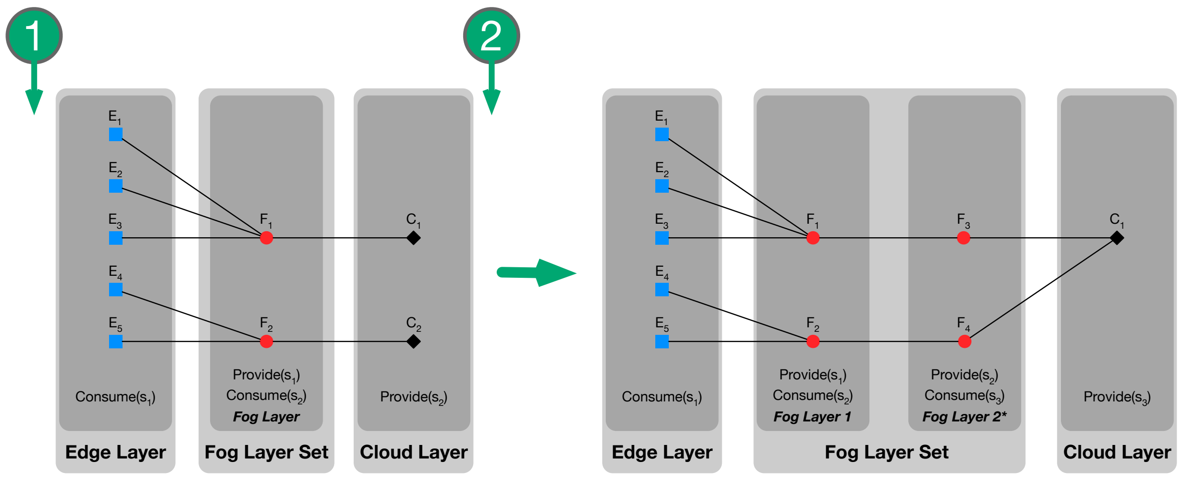


Figure 4.5: The graph shows the third layer addition which adds a new *Edge Layer* (①) or *Cloud Layer* (②) to the Fog Architecture. While this approach can also be used to transform a client-server architecture to a Fog Architecture, the given example adds a new layer on top of the *Cloud Layer* or below the *Edge Layer*. Therefore, the former *Cloud Layer* or *Edge Layer* becomes part of the *Fog Layer Set*, which is indicated by the *Fog Layer 2**, and the newly added layer becomes the new *Cloud Layer* or *Edge Layer*. In this approach, no services need to be adjusted, but the concept itself changes.

4.2.2 Stacked Fog Architectures

Adding new layers can result in conceptual changes to the proposed architecture. The most distinct change can be seen in Figure 4.3 when transitioning from a 2-layered client-server architecture to a 3-layered Fog Architecture. Although, in the example only two Fog Components are added and only one service needs to be changed, the architectural design and idea needs to be reconsidered.

Figure 4.5 shows an interesting case by adding new layers below the `Edge Layer` or on top of the `Cloud Layer`. While the required changes are even minor compared to the addition of a new layer in between the Client Layer and Server Layer as no changes to the existing services are required, the conceptual implications are bigger. The Fog Components that were previously on the Edge Layer or Cloud Layer, are moved from their layer to a Fog Layer of the Fog Layer Set. Thus, this dynamic conversion of the layer assignment can occur several times during the lifecycle, and therefore at runtime, of an application, whenever new services are introduced at the Fog Architecture's boundaries. On the other side, when layers are removed using the same approaches as for the addition, entire layers of the Fog Layer Set can be converted back to an Edge Layer or Cloud Layer.

The context, and therefore the layer assignment change, can happen even more often when different views on the Fog Architecture are considered. A *View* is a level of abstraction that is of current interest for the viewer. It can be compared to a microscope, focusing on different parts depending on the current level of zoom. We address the level of zoom by the *Abstraction Level Pointer* and the current focus range is represented by the *View* itself. Using the relation that returns the layer based on the provided integer (Definition 29), we define a *View* as shown in Definition 31.

Definition 31: Fog Architecture: View, Viewpoint, and Abstraction Level

A **View** of a Fog Architecture is a part of a Fog Architecture that consists of a specified amount of layers, that are of current interest for a stakeholder. The definition is based on the view concept introduced in SysML which provides a perspective that spans different abstractions, in our case layers [72]. Each view in SysML conforms to a viewpoint which specifies the included methods, stakeholders, and purposes. Based on the SysML view concept, our *View* is defined based on a tuple that contains natural numbers which refer to the selected layers of interest.

$$View(s) := \bigcup_{i=0}^{|s|-1} Layer(s_i)$$

with:

$s :=$ Tuple containing the numbers referring to the selected layers
$|s| =$ Amount of layers within the View $\leqslant |FogArchitecture|$
$View \subseteq FogArchitecture$

As Fog Computing highlights locality, we additionally introduce a View definition which represents a special case of the first definition. It describes a sliding window of a given size by specifying the amount of adjacent layers and the **Abstraction Level Pointer** (ALP). The ALP defines the current point of interest. For the first definition, the ALP is a set of numbers that specify the layers of interest, for the second definition, it is sufficient for the ALP to point at the topmost layer of interest, and therefore be a single number.

$$View(ALP, n) := \bigcup_{i=0}^{n-1} Layer(ALP - i)$$

with:

$ALP = AbstractionLevelPointer$
$n =$ Amount of layers within the View
$ALP - n \geqslant 0$
$View \subseteq FogArchitecture$

Each *Viewpoint* is defined as a triple that defines the list of interested stakeholders, the ALP including the amount of adjacent layers, and its purpose. Relying on our service-based layer definition, the ALP can be used to evaluate the included methods, and thus our Viewpoint definition excludes methods.

$$Viewpoint := (\{Stakeholders\},\ ALP,\ \{Purposes\})$$

To enable such abstractions, we utilize the composite pattern. Layers which are too high for the current Abstraction Level are excluded, while lower layers are summarized into their higher level composite: Each Fog Component X that offers at least one service S, respectively, Fog Nodes and Cloud Devices, is part of a composite pattern. The Fog Component X is therefore the composite while every Fog Component Y that consumes the service S is a leaf. To enable the View concept without loosing data, the leaves are combined into the composite establishing one new Fog Component which is the new leaf of the layer above.

An example for a *View* on the infrastructure example, which we used to describe the problems in Chapter 1, is shown in Figure 4.6. It represents the $View(2,3)$ of the Fog Architecture. ① shows the current Abstraction Level of 2 and the according scale from low (0) to high (5). The values of the Abstraction Level are discrete values relating to all available layers. The *View* or sliding window (②) contains the lowest three layers, highlighting local Fog Components and their interaction at the *Edge* with the *Crossings*, and their according *Streets*. Everything on top of the *Streets* has an Abstraction Level which is too high as indicated by ①.

Although this approach can potentially separate the Fog Architecture in several unconnected subgraphs, it allows to focus on specific aspects of the Fog Architecture. This is especially helpful in Fog Architectures with many Fog Components.

Additionally, Views provide another insight into the Fog Architectures. When looking at two adjacent layers, it shows the service providers on the higher level and the service consumers on the lower level. We use this approach in xFogStar in Chapter 5 to find the best communication partner for Fog Components on adjacent layers.

Another observation relates to the three adjacent layers as shown with the sliding window in the example in Figure 4.6. The subgraph describes a Fog Architecture itself with the lowest layer (⑤) being physically closest to the next layer (④) which in turn is closest to ③. This is also indicated with the three red boxes next to ③, ④, and ⑤ which show this *Cloud*, *Fog*, and *Edge* structure (CFE-STRUCTURE). The *Street Fog Component*, represented by a traffic light, can be seen as the Cloud Layer which aggregates all data from the individual *Crossings*, which in turn aggregate data from the individual traffic participants at the *Edge*.

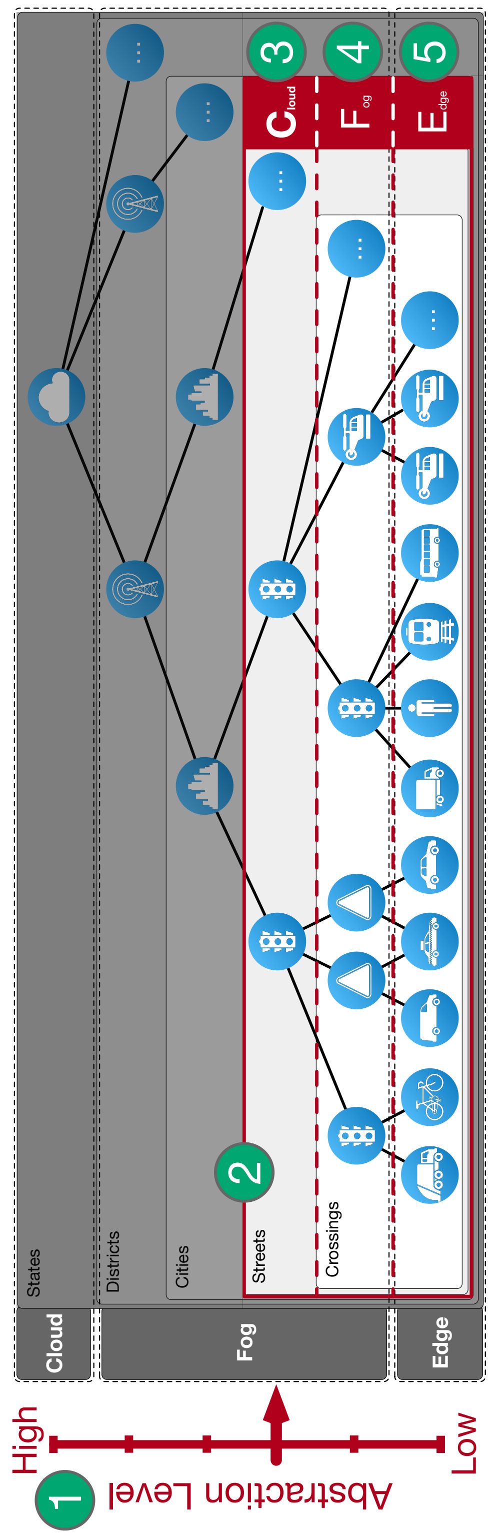


Figure 4.6: The diagram shows a an example for a View on the infrastructure example from Chapter 2. ① shows the Abstraction Level scale and the Abstraction Level Pointer. The Abstraction Level is a discrete value relating to the available layers. With the focus on *Fog Components* at the edge, the View in the red box (②) highlights the *Edge Layer* (⑤) and the first two layers of the *Fog Layer* (③, ④). Higher level layers are abstracted away.

This idea is further investigated in Figure 4.7, in which we show all different 3-layer views with only a single Fog Layer in the Fog Layer Set. On the lowest layers (①), the view highlights the interactions between individual components, the current *Crossing* they are at, and the containing *Street*. Based on the locality, Fog Components deployed as *Crossings* are the first point of contact for individual vehicles and can decide which vehicle can pass the *Crossing*. On the next higher level, the *Streets* receive information from all *Crossings* to represent higher-level aspects, such as the traffic density.

From the perspective of a Fog Component on the *City* layer, shown in ②, individual vehicles might already be too much detail, if they want to decide which streets have the most traffic volume and need alternative routes. While the amount of vehicles and the flow is of interest, it is irrelevant whether the vehicle is a car, bus, or train.

The next higher view (③), takes the position of *Districts*. With the information of *Streets* and *Cities*, they can e. g. see the traffic density on roads outside of cities; again the information of *Crossings* is too low level at this point.

Finally, the view highlighted by ④ shows the highest level Fog Architecture which puts its focus on the interaction between *Cities* and *States*. For this architecture, individual streets within a city are of no interest, as it would rise the complexity level.

In general, the more complex the application, the more layers are within a Fog Architecture. Even in the presented infrastructure example, several layers can be added. *States* are by far not the highest instance and the *Vehicles* can also consist of individual layers of Fog Components such as sensors. Also in between the presented layers, other layers could be added, e. g. a *Borough* layer to gather different parts of *Cities*.

Using this concept, we can redefine a Fog Architecture to consist of several 3-layered Fog Architectures.

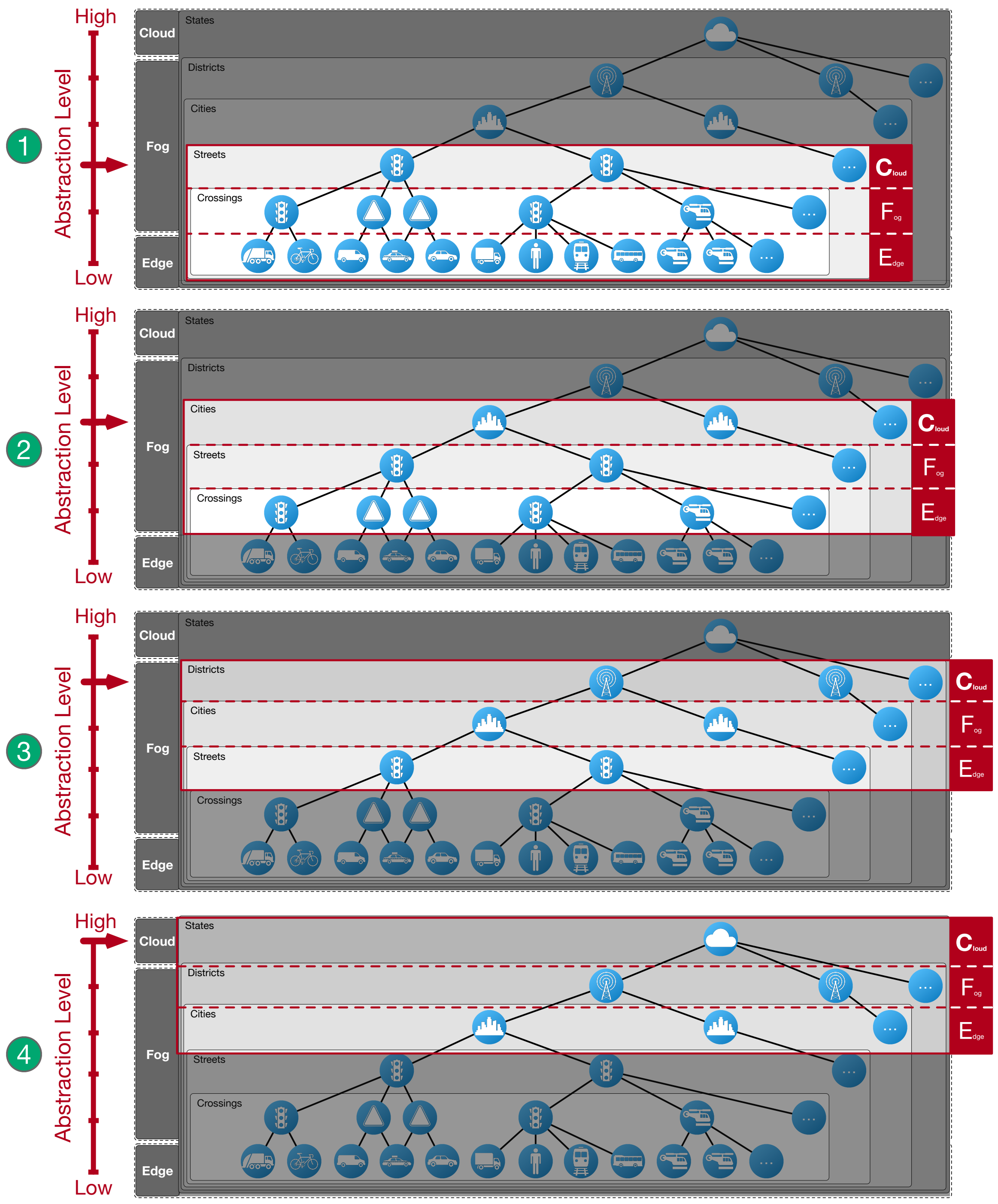


Figure 4.7: Using the *View* concept, we show all 3-layered Fog Architectures for the infrastructure example from Figure 1.2. ① shows the *View* which focuses on the *Edge Layer* and the two adjacent layers above. The *View* in ② includes the layer describing *Cities*, but abstracts the *Edge Layer* to reduce complexity and to allow to focus on other aspects of the Fog Architecture. Moving one layer higher (③), the *View* abstracts from *Crossings*, but includes *Districts*. This allows, e. g., to see traffic density on cross country roads. Finally, in ④, the *View* includes *States* but abstracts away individual *Streets*.

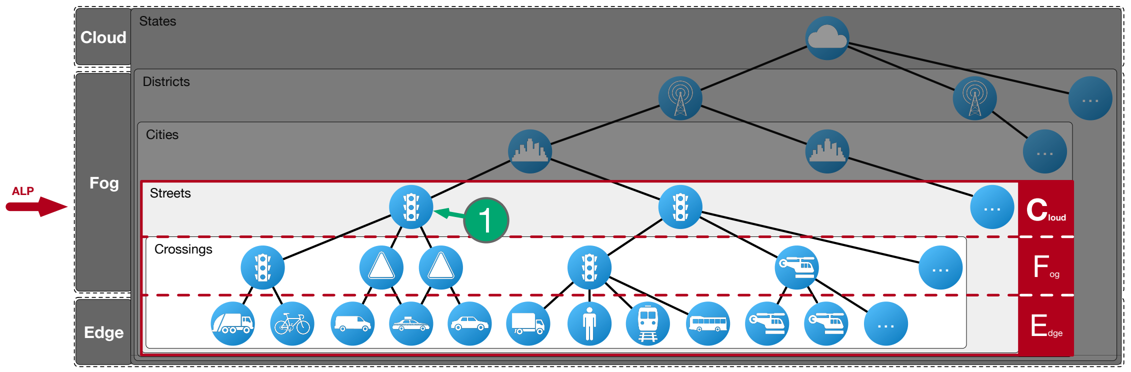
4.2.3 Dynamic Type Change

In this section, we address the implications that the *View* concept has on the individual Fog Components of a Fog Architecture on the different layers that we described in Section 4.1.2. Figure 4.8 consists of three subfigures showing three 3-layer Views of the infrastructure example. For simplicity, we only show the *Abstraction Level Pointer* (ALP) to indicate the Views focus and do not show the Abstraction Level scale. The first subfigure shows the View that highlights the lowest three layers, respectively, the `Edge Layer`, `Fog Layer 1`, and `Fog Layer 2`. In the second subfigure, the View shows *Fog Layer 1* and *Fog Layer 2*, while leaving out the `Edge Layer`, but therefore adding the `Fog Layer 3`. The last View adds one additional layer on top, but leaves out `Fog Layer 1`. Each of these Views represents a Fog Architecture itself as indicated by the `Cloud`-, `Fog`-, and `Edge`-label on the right.

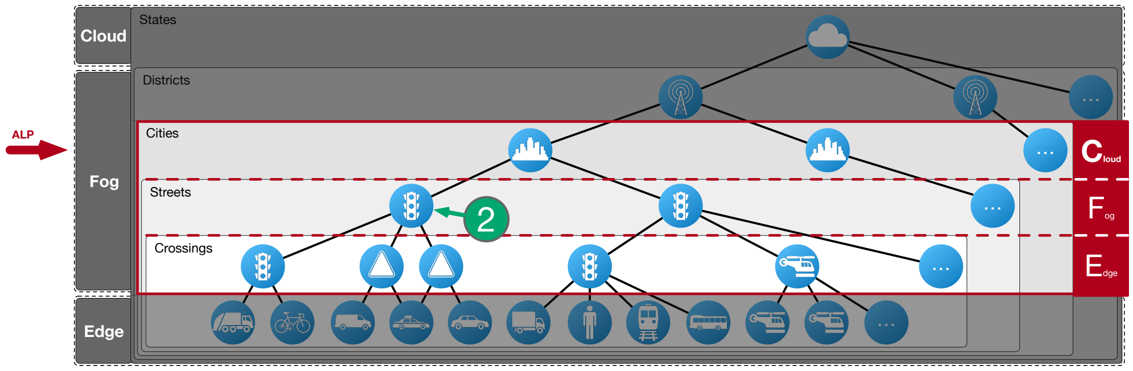
For each subfigure, we take a look at the highlighted `Fog Component` (①, ②, ③). This Fog Component is the same Fog Component in the different provided Views. Depending on the View, it changes its layer assignment. While in ①, the Fog Component is assigned to the `Cloud Layer`, for ②, it is on the `Fog Layer` and for the ③, the Fog Component is on the `Edge Layer`. Therefore, this Fog Component is either considered a Cloud Device, Fog Node, or Edge Device.

To address this issue of dynamic type changes, we use the Fog Component definition shown in Section 3.2.1, which introduced a Fog Type on MoF M3. This definition of the Fog Type on M3 with the three subclasses Edge Device, Fog Node, and Cloud Devices allows us to use the term Fog Component in our meta models. Therefore, the Fog Component dynamically switches its type without a need to redefine our introduced concepts in Chapter 3. Our layer definitions introduced in Section 4.1.2 still hold true and provide us with the layering for the overarching Fog Architecture.

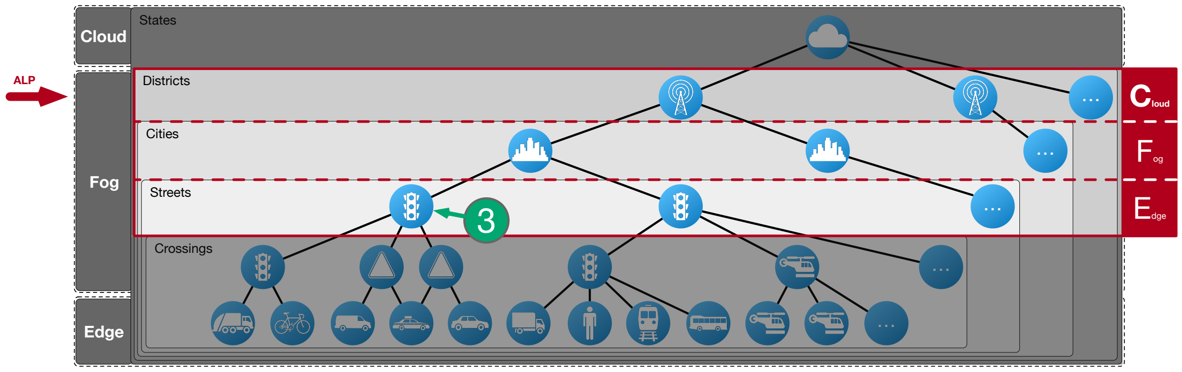
The relation between the original layers and the types that Fog Components on those layers can dynamically adapt to based on different Views is stated in Definition 32.



(a) The *View* in this subfigure is set to *Abstraction Level* 2. The *View* contains three layers: The *Edge Layer*, *Fog Layer 1*, and *Fog Layer 2*.



(b) The *View* in this subfigure is set to *Abstraction Level* 3. The *View* contains three layers: *Fog Layer 1*, *Fog Layer 2*, and *Fog Layer 3*.



(c) The *View* in this subfigure is set to *Abstraction Level* 4. The *View* contains the three layers: *Fog Layer 2*, *Fog Layer 3*, and *Fog Layer 4*.

Figure 4.8: The example shows three 3-layer Views on the infrastructure example introduced in Figure 1.2. For simplicity, we only show the Abstraction Level Pointer instead of the entire Abstraction Level scale. It highlights the same *Fog Component* in each subfigure (①, ②, ③), while changing the *Abstraction Level* of the *View*. According to the current *View*, this Fog Component changes the layer it is considered to be on from *Cloud Layer* in ① to *Fog Layer* in ②, and to *Edge Layer* in ③. Thus, depending on the *View*, the Fog Component changes its *Fog Type*.

Definition 32: Layers and dynamic Types

The following rules describe the relation between the *Fog Components* of the original layers of a Fog Architecture and the type that they can dynamically change to:

1. *Fog Components* in the *Cloud Layer* can only be considered *Cloud Devices*, if they are included within a View of a Fog Architecture.
2. *Fog Components* in the *Fog Layer Set*, respectively in any *Fog Layer #*, can dynamically change their type. We start counting *Fog Layers* on the lowest level with 1 going up to *Fog Layer n* which is the layer in direct contact with the *Cloud Layer*:
    1. *Fog Components* in *Fog Layer n*, with *n* being the total amount of *Fog Layers*, can dynamically change their type between *Fog Node* and *Cloud Device*.
    2. *Fog Components* in the layers between 1 and *n* can dynamically change between any type.
    3. *Fog Components* in *Fog Layer 1* can dynamically change their type between a *Fog Node* and *Edge Device* depending on the View.
3. *Fog Components* in the *Edge Layer* can only be considered *Edge Devices*, if they are included in a View of the Fog Architecture.

Chapter 5

xFogStar: A Workflow for Service Provider Selection

"The fact that in daily life people are often able to choose their interaction partners can be considered as an endogenous regrouping device, which is also an effective way to escape exploitation. Indeed, people frequently change or quit relationships with individuals who are not fulfilling the expected cooperative standards and look out for better opportunities, even if it involves substantial costs."

— GIORGIO CORICELLI [33]

Based on xFog and accordingly xFogCore, and xFogPlus many new concepts can be established in dynamically, scalable Fog Architecture. In this chapter, we introduce xFogStar, one such concept that focuses on the relation between service consumers and service providers, the players suggested by Bonomi et al. [18]. As already addressed in Section 4.2.2, this relation is highlighted by a 2-layered View for a service on adjacent layers. Figure 5.1 shows the infrastructure example from Chapter 1, but uses the View concept with two layers to focus on the Edge Layer and the first Fog Layer.

This View is particularly interesting if new Edge Devices should be dynamically added to the Fog Architecture using the concepts of xFogPlus (Chapter 4). As we want to address the relation between service consumers and service providers, in this chapter, we do not focus on the requirements for the added component to be considered a Fog Component or the involved sets, but rather focus on the case that multiple service providers are in the Fog Component's Fog Horizon. Figure 5.2 shows a Fog Component *Car* in the infrastructure setup with four potential service providers in its Fog Horizon that offer the requested service S. Accordingly, we can limit the Fog Horizon to that service.

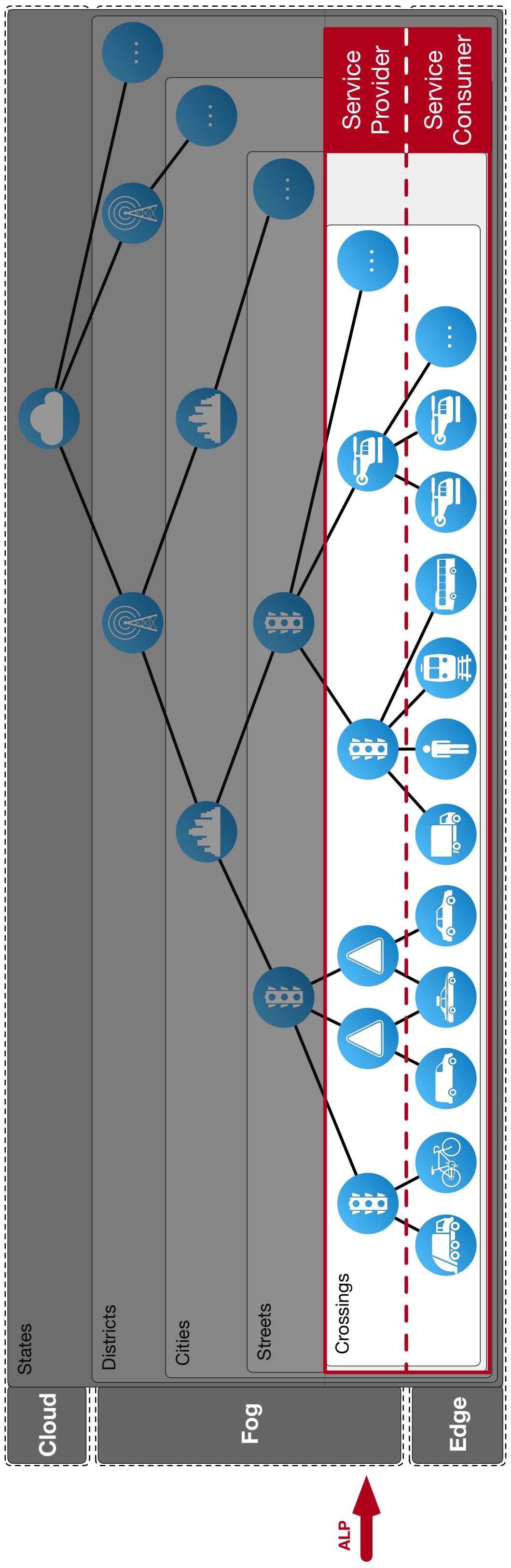


Figure 5.1: 2-layered View in the infrastructure example from Chapter 1. The View highlights the *Service Provider* and *Service Consumer* relation which is given by adjacent layers.

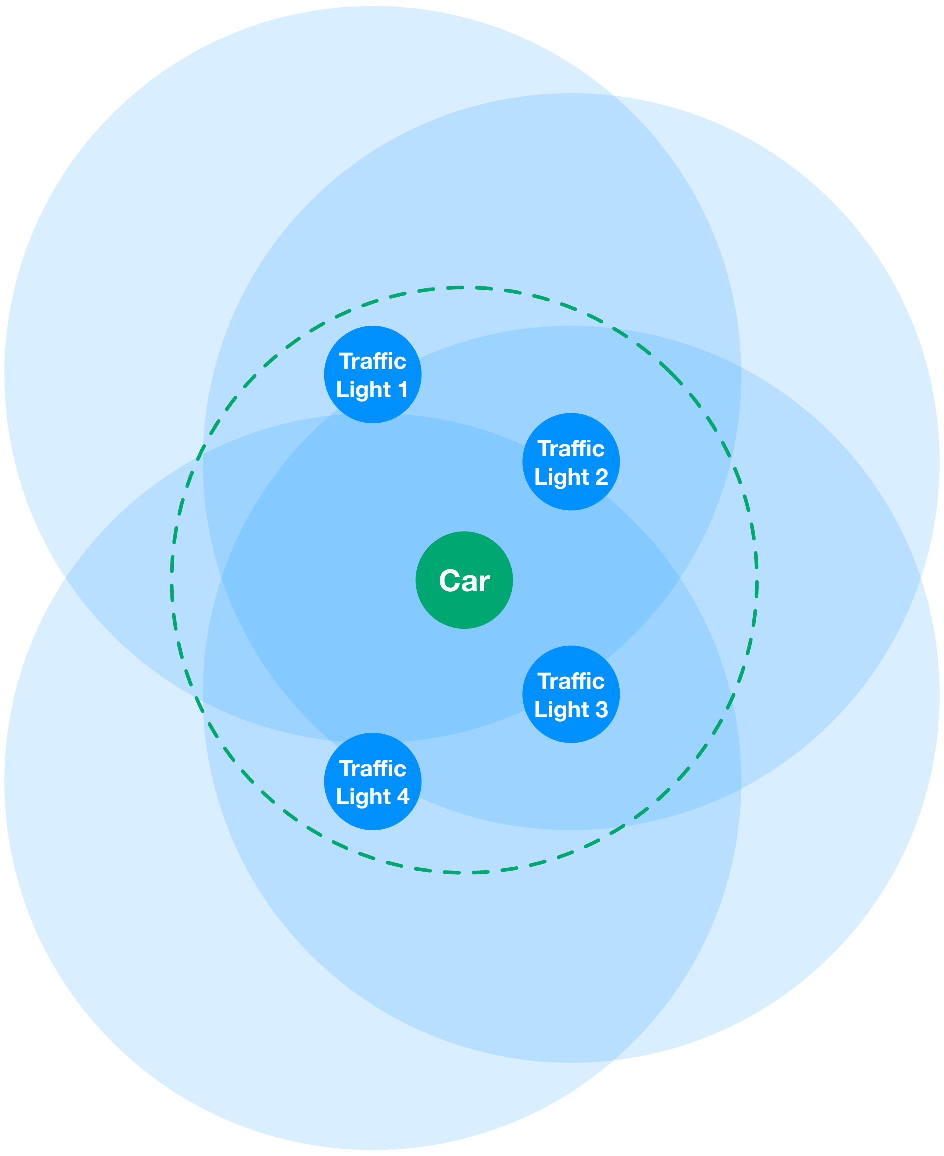


Figure 5.2: Fog Visibility of a new Fog Component *Car* which is added to the Fog Architecture as shown in Figure 5.1. Its Fog Visibility is represented as a green circle with a dashed border. The Fog Visibilities of the service providers (traffic lights) are shown as blue circles around their label.

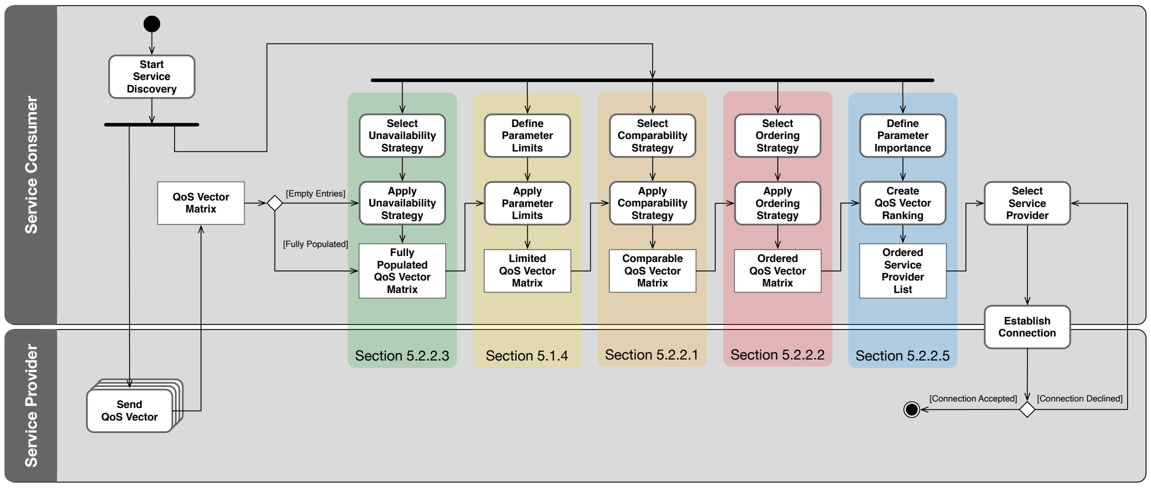


Figure 5.3: The Workflow for Service Provider Selection (UML Activity Diagram). Each service provider selection starts with the discovery of the service providers which answer with a QoS vector containing their QoS parameters. The following process is split into six sub-workflows: Unavailability (*green*), Limits (*yellow*), Comparability (*orange*), Ordering (*red*), Parameter Importance (*blue*), and Service Provider Selection.

For each service provider, the blue circles around the label indicate their Fog Visibilities and the green circle with the dashed border shows the Fog Visibility of the Fog Component which should be added.

As the name suggests, this chapter focuses on the workflow that describes how to select the best fitting service provider in the case that multiple service providers are in the FogHorizon$_S$ of the new Fog Component. Thus, addressing functional requirements 7-11 and non-functional requirements 2, 3, and 7. To describe properties of the service provider and its service, but also the needs of the service consumers, we use QoS parameters and the according vector as introduced in Section 2.1.3. The workflow, represented as an UML activity diagram, is shown in Figure 5.3.

The workflow starts with the discovery of the available service providers that offer a requested service (FogHorizon$_S$). Every service provider which offers this service `Send QoS Vector` as a response to the Fog Component that requests the service. We introduce a non-exhaustive list of potential QoS parameters and their dependencies in Section 5.1.1. The service consumer gathers the QoS vectors in a QoS matrix. In case the matrix is not fully populated (shown in green), the service consumer has to `Select Unavailability Strategy` that decides on how to treat missing parameters which we address in Section 5.2.2. In yellow, it shows the second step, the `Define Parameter Limits` activity which we describe in Section 5.1.4. The remaining steps are described in Section 5.2.2: The third step, shown in orange, is the `Select[ion of a] Comparability Strategy`. It defines how different parameters, e. g., costs and times, are compared with each other.

Fourth, shown in red, the service consumer has to `Select Ordering Strategy` that allows to compare parameters that prefer high values with parameters that prefer low values. Shown in blue, we address that service consumers might value parameters with different importance. Finally, the service consumer receives an `Ordered Service Provider List` in which the best fitting service provider is the first entry.

5.1 Quality of Service Parameters

In this section, we introduce the QoS parameters that are of interest in the Fog Computing context. In Section 5.1.1, we define the used parameters with their according value types and ranges. In the following Section 5.1.2 and Section 5.1.3, we introduce categories and a level concept to classify the QoS parameters according to their dependencies. Finally, in Section 5.1.4, we introduce the possibility to assign weights for each parameter to define their importance for the service consumer in the current application's context and apply limits which define a value range in that the parameters should be in.

5.1.1 Definition

To find the best fitting partner to communicate with in a given Fog Architecture, we introduce an extensive list of potentially interesting parameters for service consumers as well as service providers. Figure 5.4 shows the list of parameters in an UML diagram. While the parameters on the left side are individual parameters, the parameters on the right side always show up in four different versions. Those versions are related to `Execution`, `Storage`, `Memory`, and `Network` as shown in Figure 5.5. This diagram provides the four versions by defining a new type on MoF Level 2 of which the different parameters are instantiations. For instance the parameter `Time` will occur in the variations *ExecutionTime*, *StorageTime*, *MemoryTime*, and *NetworkTime*. This allows a more specific tracing of the individual parameters values.

Based on this categorization, the following list describes the different parameters which can be addressed.

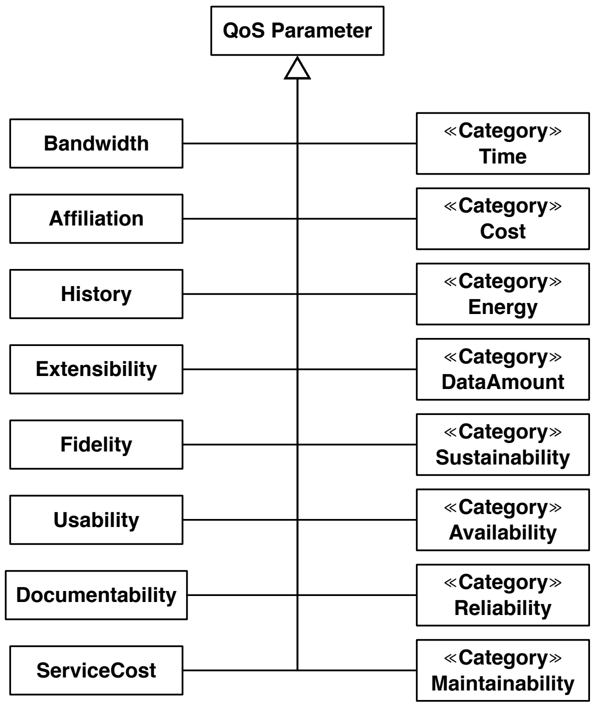


Figure 5.4: Overview of all introduced parameters as subclasses of the *QoS Parameter*. Each parameter of type ≪Category≫ has additional subcategories *Execution, Storage*, *Memory*, and *Network*.

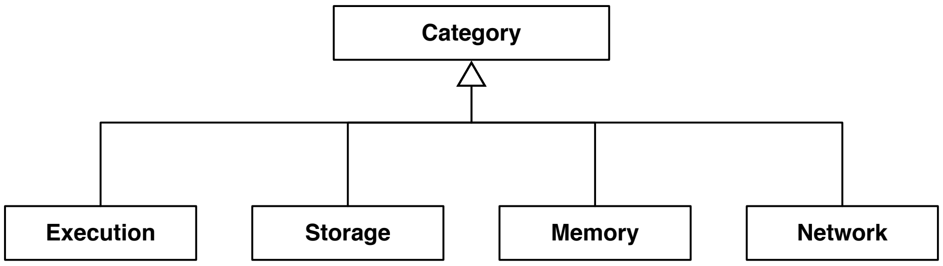


Figure 5.5: Definition of the *Category* type with its subclasses *Execution*, *Storage*, *Memory*, and *Network*. Providing the type definition allows a representation of the parameters with less complexity.

Time (T)

The *Time* parameter can be separated into four individual parameters: *ExecutionTime*, *StorageTime*, *MemoryTime*, and *NetworkTime*. Each of them covers a different aspect of the time that is required to receive an answer from a service provider for a given service. The *ExecutionTime* is the time required to execute the requested service on the service provider. It is linked to the computational power a service provider is capable of, but relates to a given service. Other sources often refer to this parameter as the computation time [91], which we adjusted to create a consistent taxonomy. The *StorageTime* is defined as the time needed to store data for the requested service on the service providers machine. The *MemoryTime* is the time needed to keep the data for the given task in memory of the service provider. While both of these times have rather limited use for a service consumer, they provide indirect hints to the execution power of the service providers machine as well as their connectivity to other components in the network. Finally, the *NetworkTime* is comprised of the *CommunicationTime* and the *ForwardingTime*. It sums up the total time needed for the network to transfer the request and the answer from the service consumer to the service provider and vise versa. Therefore, the *CommunicationTime* takes the major part of the needed time, measuring the time it takes to send data from one communication partner in the network to another one within their Fog Visibility. The *ForwardingTime* is the amount of time needed by a hop to forward the data to the next communication partner. Thus, the *ForwardingTime* only appears in the cases of transitive networks, in our case Fog Reachabilities and Fog Sets as defined in Chapter 3. The Time is measured in milliseconds (ms) $\rightarrow \mathrm{T} \in \mathbb{N}_0$.

Cost (C)

Also listed separately in Figure 5.4, the cost consists of 5 parts: *ExecutionCost*, *StorageCost*, *MemoryCost*, *NetworkCost*, and *ServiceCost*. The *ExecutionCost*, *StorageCost*, and *MemoryCost* define the costs the service providers have to pay to run a service on their Fog Component. The *NetworkCost* consists of the costs for using the given network. Therefore, each Fog Component can raise costs for forwarding requests, but also the network owner can take its share. Finally, the *ServiceCost* is the price the service provider adds to his given costs for offering and running the requested service. Hassan et al. use the term execution cost as the sum of all costs that are required for a successful execution of the service [64]. In our case, those costs reflect *ExecutionCost*, *StorageCost*, *MemoryCost*, and *Service Cost*. While the costs are taken in consideration and are a key parameter for an open market and several service providers offering the same service, we do not address payment. The given concepts do not cover how the service consumer pays the service provider and all network participants in between. The costs are measured in U.S. dollars $\rightarrow \mathrm{C} \in \mathbb{Q}_0^+$.

Energy (E)

The *Energy* parameter is particularly important in the context of mobile or battery powered devices. From the perspective of a service provider, they intend to maximize the profit with the limited energy in mind. Service consumers prefer service providers who can offer a service for less energy which reduces the environmental impact. The Energy parameter is split in the four categories introduced in Figure 5.5 with their respective energy consumptions. Energy consumption as a noteworthy parameter was already addressed several times as for instance in the work by Deng et al. [39] or Faruque et al. [3]. Given the two different perspectives on Energy by the service consumer and provider, *Energy* can be defined in two different ways. While service consumers are interested in the absolute value of energy consumption compared to other service providers, service providers care about the energy consumption for offering this service in comparison to the available remaining energy. Thus, *Energy* is either given in watts W providing absolute values $\rightarrow \mathrm{E} \in \mathbb{Q}_0^+$ or as a percentage value comparing the needed energy against the available remaining energy $\rightarrow \mathrm{E} \in [0, 100]$. In this chapter, we use the absolute value.

Sustainability (S)

While the term *Sustainability* has several aspects and is mostly considered as "meeting fundamental human needs while preserving the life-support system of planet Earth" as described by UN Secretary General Kofi Annan in 2000 [7] and picked up by Kates et al. in "Sustainability Sciences" [78], in the context of Fog Architectures and parameters, we base *Sustainability* on the relation between renewable energy and non-renewable energy of the energy needed to deliver the given service to the service consumer. *Sustainability* can be split up in the four introduced categories relating to each different energy consumption. The *Sustainability* is measured in percentage of renewable energy $\rightarrow \mathrm{S} \in [0, 100]$.

Data Amount (DA)

This parameter is the amount of data which has to be stored or transferred over the network whenever a service is requested. This *DataAmount* gives an idea to which extend the bandwidth between the service consumer and service provider will be used for a service or how much data is exchanged on the machine itself. Storage was addressed by Bonomi et al. [17] or by Yi et al. [156]. While they address storage as the storage amount needed on the computational unit, in our taxonomy, *DataAmount* has the previously defined four categories giving it a broader meaning. The *DataAmount* is measured in bytes $\rightarrow \mathrm{DA} \in \mathbb{N}_0$.

Availability (A)

The *Availability* is split up in four categories *ExecutionAvailability*, *StorageAvailability*, *MemoryAvailability*, and *NetworkAvailability*. The availabilities for execution, storage, and memory are based on the hardware of the device the service provider uses as well as the other services offered by this service provider. Thus, a service provider's availability decreases the more services are requested. Depending of the priorities of the service consumers, this results in a natural balancing of services on service providers. The *Availability* is measured in percentage of uptime (%) $\rightarrow \mathrm{A} \in [0, 100]$.

Reliability (R)

Reliability is defined as the mean-time-between-failures. It is split up in the four categories and allows the service consumer to specify uptimes a service has to provide. While the service provider ensures the reliabilities of the execution, storage, and memory, the network reliability consists of the access of the service provider to the network, the access of the service consumer, and all Fog Components in between making it difficult to measure. Nevertheless, many service consumers rely on the provided services making this parameter particularly important. The *Reliability* is measured in milliseconds (ms) $\rightarrow \mathrm{R} \in \mathbb{N}_0$

Maintainability (M)

The *Maintainability* is defined as the mean-time until the requested service is back at an operable state after the occurrence of a failure. Especially in combination with Reliability, *Maintainability* allows the service consumer to make the prediction how often and how long a service offered by a service provider is unavailable. This parameter is in particular important in the context of software and received a lot of attention in the last years. Already in 1993, Li and Henry tried to connect several object-oriented programming metrics to the system's maintainability [86]. While their definition focused on the maintenance effort based on different metrics, we measure *Maintainability* by the time it takes for a system to be operable after a failure in milliseconds (ms) $\rightarrow \mathrm{M} \in \mathbb{N}_0$.

Bandwidth (B)

The *Bandwidth* is the first quality of service parameter which is not split up in the four categories as shown in Figure 5.5. As the name suggests, the *Bandwidth* is the maximum capacity of data which can be transferred between the service consumer and service provider. This metric was already used by Li et al. [156] and is interesting in applications that depend on a specific minimum bandwidth. In the gaming domain, those applications are quite common as for instance shown in [89, 159]. As shown by Pantel et al. [105] and Jarschel et al. [73] a delay of 100ms, which can be based on *Bandwidth* limitations, will already cause issues. This parameter is measured in MBit/s $\rightarrow \mathrm{B} \in \mathbb{Q}_0^+$.

Affiliation (AFF)

This parameter was used by Lin and Shen [89] to allow users of the same social group to connect to the same gaming service providers as they will probably interact with each other within the game. While they called the parameter "social network based server assignment", we refer to it as *Affiliation* to allow a broader view on connectivity between service provider and consumer on top of the social network use case. Therefore, the *Affiliation* indicates the belonging of the service consumer and service provider to the same social groups. We define the parameter as a boolean value that indicates if service provider and service consumer belong to the same group, but the parameter can easily be extended to name group belongings, and thus allow service consumers of the same group to closely interact using the same service provider $\rightarrow \mathrm{Aff} \in \mathbb{B}$. This parameter allows service consumers to prioritize service providers within their own social group, as for instance prioritizing money exchange within their own company than requesting services from outside.

History (H)

The *History* describes previous experiences between the service consumer and service provider. The better the previous experience of the service consumer with the offered service of a service provider, the higher the value. This approach was used by Lin and Shen [89] as well as by Mahmud et al. [91] and mirrors the idea of the Net Promoter Score as introduced by Reichheld [114]. While the Net Promoter Score, or NPS for short, is used from a service provider perspective comparing how many customers or service consumers would recommend the service provider to a friend or colleague, our definition switches to the service consumer view. It indicates the likeliness to use the service offered by the service provider in case that other service providers are available. The *History* parameter is, same as the NPS, measured in percentage $\rightarrow \mathrm{H} \in [0, 100]$.

Locality (L)

Based on the work of Intharawijitr et al. in their analysis on 5G networks [71], one important aspect of Fog Architectures is the distance of the service providers from its consumers which is indicated by the amount of hops between them and the delay that those hops produce. While the delay is already considered by the *NetworkTime*, we name the amount of hops *Locality*. The *Locality* in combination with the *ForwardingTime* and *CommunicationTime* provides the total communication delay that is introduced by using the specific Fog Path (Section 3.4.1) to reach the specified service of the service provider. If the service is provided on the same Fog Component as it is used on, the *Locality* is 0, if the service is provided in the immediate Fog Horizon of the service consumer, the *Locality* is 1 and increased by 1 for every hop in between the consumer and provider. Therefore, *Locality* is defined as a natural number $\rightarrow \mathrm{L} \in \mathbb{N}_0$.

Extensibility (Ex)

The *Extensibility* focuses on the interaction with humans during the discovery process. If a service is extended for other use, such as machine learning models, a human might prioritize a service with a high value in this area over a highly specialized service to reduce the amount of discoveries they have to go through for different services. Additionally, this parameter can be defined as the amount of different services a service provider offers, increasing the chance that the service consumer does not have to perform another discovery. Depending on the selected definition, *Extensibility* is either measured in percentage indicating the scale of specialization of a given service or as the number of offered services by a service provider $\rightarrow \mathrm{Ex} \in [0, 100] \vee \mathrm{Ex} \in \mathbb{N}$. In the remaining of this dissertation, we use the services count.

Fidelity (F)

The *Fidelity* indicates the accuracy of the provided service against its specifications. In a highly automated system, the accuracy needs to be close to 100%, but in systems including humans, lower levels might be acceptable. While the *Fidelity* is an interesting parameter to take into consideration, it is usually not available during discovery time, and therefore not applicable in the discovery process. Nevertheless, we will include it in the diagrams for the sake of completeness. Additionally, *Fidelity* can be used to define the History parameter as introduced before. If a service consumer is receiving the service from a service provider as specified, and therefore as expected by the service consumer, the service consumer is more likely to request a service from this service provider later on. *Fidelity* is measured in percentage % $\rightarrow \mathrm{F} \in [0, 100]$.

Usability (U)

The *Usability* measures the quality of the service based on user-friendliness. Therefore, it specifies how easy the service and the results could be integrated in a context. The *Usability* shows the same issue as the fidelity: While it would be interesting to know during discovery, only experiences with the service and the according service provider can state this parameter for a service consumer. Despite that, the *Usability* is still listed here, as it provides the second aspect that is needed to define the previous history with the service provider or the service itself in particular. Sauro and Kindlund present a method to get a single, standardized, and summated usability parameter for all usability aspects which can be used as the send parameter [121]. We measure *Usability* in percentage % $\rightarrow \text{U} \in [0, 100]$.

Documentability (D)

Finally, the last parameter addressed is the *Documentability*. It is more helpful for humans developing service consumers and specifies how well the requested service is documented. While this parameter is redundant in fully automated architectures, human centric systems and domains highly profit from this parameter. *Documentability* addresses two parts: The amount of documentation provided for the service and the understandability of it. For simplicity, the *Documentability* is measured in percentage, but could also be addressed with more elaborate calculations as done by Sauro and Kindlund for Usability $\rightarrow \text{D} \in [0, 100]$.

5.1.2 Categorization

The parameters can be categorized based on their relation to the four entities that have a direct impact on some or all of the parameters: Service consumers, service providers, networks, and services. Using the mathematical definition of dependency, each parameter is either dependent or independent of the given class as presented in Figure 5.6. Therefore, each of the four presented models comprises all described parameters from Section 5.1, but they differ in their distribution.

In order to get a more compact representation, we reduce each of those models to only include the dependent subclasses and fuse those representations into a single model showing the four dependency classes: service consumer dependent, service provider dependent, service dependent, and network dependent. Figure 5.7 presents this idea. While this simplification would exclude parameters which are independent of all four classes, we argue that these four classes comprise all relevant parameters for any Fog Architecture creating a complete set. The mapping of the parameters introduced in the previous section is shown in the following section on parameter levels. In the following, the remaining four classes are referred to as *Dependency Classes*.

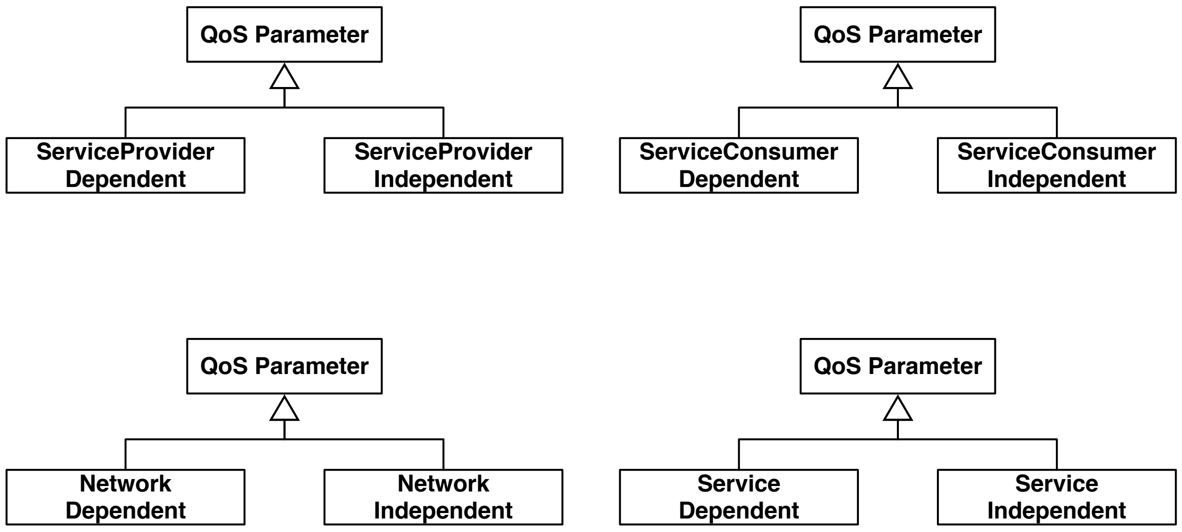


Figure 5.6: Classification of the *QoS Parameters* based on the mathematical idea of dependency and independency based on the four classes *Service Provider*, *Service Consumer*, *Network*, and *Service*. These dependencies are used to define dependency classes for the *QoS Parameters* and categorize them on similarities.

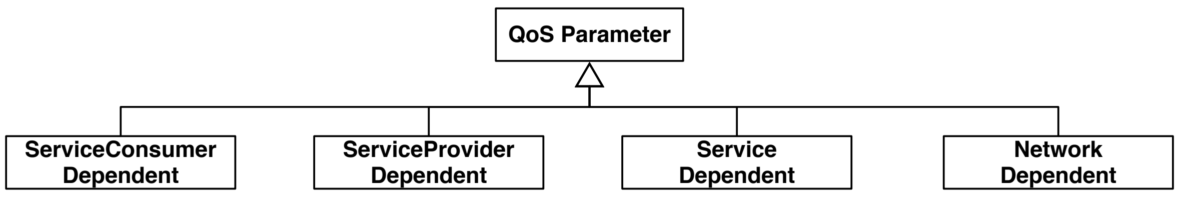


Figure 5.7: Classification of the *QoS Parameters* from Figure 5.6 reduced to the dependent classes. The independent cases are removed for simplicity, as those classes do not have any influence on the *QoS Parameters*.

Table 5.1: Dependency levels of the quality of service parameters in a comprised form. It reflects the levels introduced in Figure 5.8.

Level 1	**Level 2**	**Level 3**	**Level 4**
Sustainability	History	Fidelity	
Availability	Affiliation	Usability	
Reliability	Time	Locality	
Maintainability	Cost		
DataAmount	Energy		
Extensibility	Documentability		
Bandwidth	Service Cost		

5.1.3 Level

Figure 5.8 presents the categorization of each parameter according to its dependency classes introduced in Figure 5.7. Additionally, it introduces so-called *Levels*. These levels represent the complexity of the parameters based on the amount of dependencies they have to the four dependency classes. While parameters on level 1 are only dependent on one single dependency class, parameters on level 4 depend on all four dependency classes. In Figure 5.8, this relation is represented using multiple inheritance.

Additionally, the diagram introduces a color coding for the four parameter categories as introduced in the beginning of Section 5.1: Execution, storage, memory, and network. Table 5.1 shows a comprised version of the parameter-level relation.

While we add several parameters in the groups of level 1, 2, and 3, we did not find any parameters for level 4. As we do not claim our list of parameters to be exhaustive, additional parameters can be added to the dependencies as well as the level categories. Application domains might also add or remove parameters based on their importance.

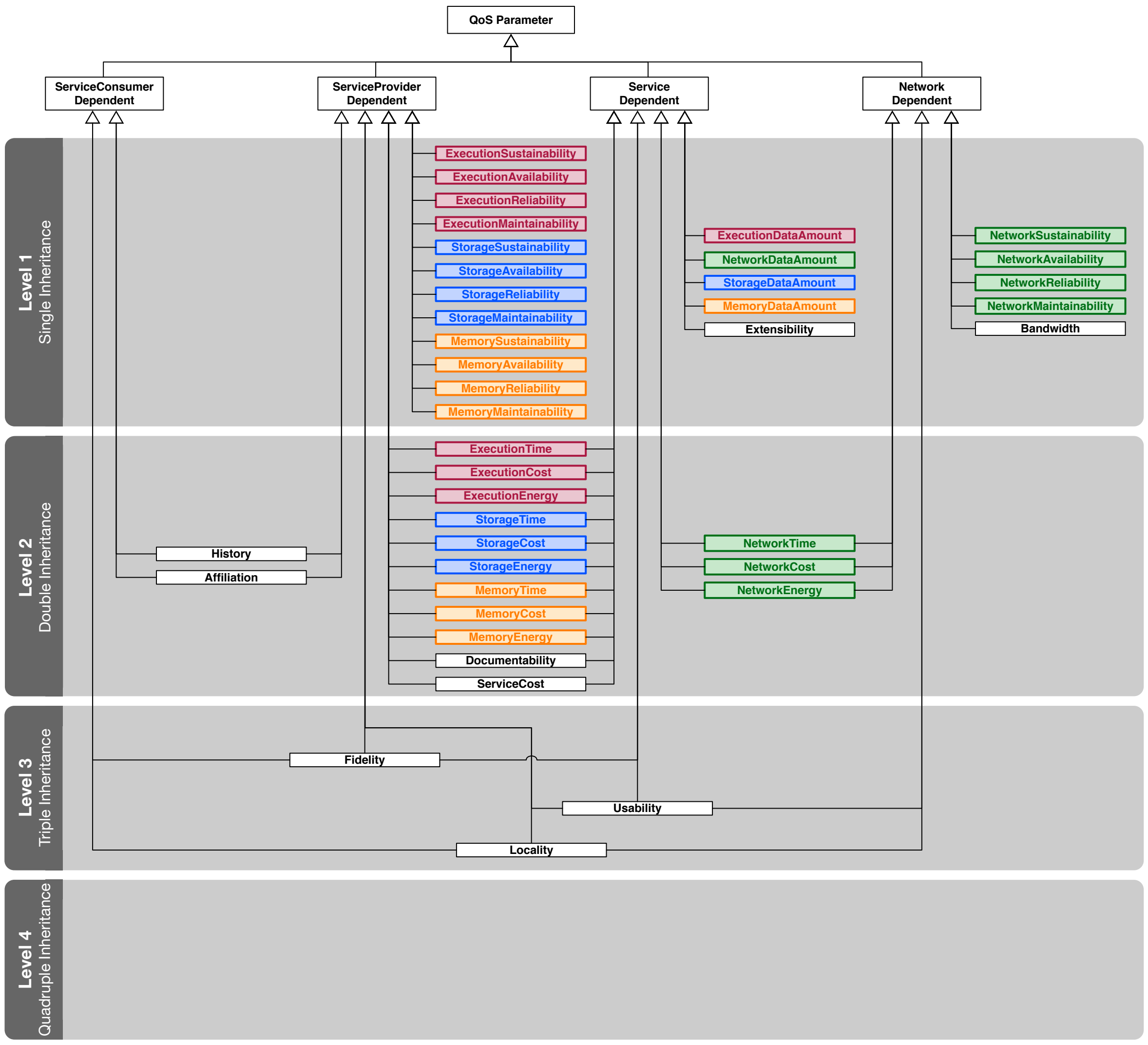


Figure 5.8: Dependencies between the entire list of quality of service parameters and the four entities: Service Provider, Service Consumer, the Service itself & Network. Those dependencies are used to group the parameters into different levels of complexity based on single, double, triple and quadruple inheritance. Additionally, a color coding is provided to group parameters that belong together. Parameters highlighted in *red* are execution specific, *blue* parameters are storage specific, *yellow* parameters are memory specific, and *green* parameters network specific.

5.1.4 Prioritization

Given the extensive list of parameters, service consumers need a way to specify which parameters are important to them and to which extend. These priorities are on the one side related to the application domain the service is used in, and on the other side based on the application itself. An application in the medical domain might, e. g., rate the overall response time, reliability, and availability higher than the energy consumption or the consumed storage. Especially in emergency rooms, having a service available right now is more important than who is providing the service. An application in an administrative context might look closer on the costs and availability instead of having the fastest response time. To integrate this idea into quality of service discovery, we allow service consumers to specify individual weights for the parameters, scaling their importance, and allowing them to set limits for the parameters that should not be exceeded.

The individual *Weights* for each quality of service parameter are multipliers scaling the importance of a parameter. Therefore, all weights $\alpha, \beta, ..., \xi$ are in the range $[0, 1] \in \mathbb{Q}$. Weights allow service consumers to specify their preferences based on the application they are using and allow the owner of the Fog Architecture to specify which parameters are considered most important in the given problem domain.

Equation 8: Weights

$$\overrightarrow{W} = (\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta, \iota, \kappa, ...)^T$$

Table 5.2 shows the assignment of parameters to weights including the parameter categories Execution, Storage, Memory, and Network. This allows the service consumer to specify the priority of each parameter individually.

While individual weights allow the most customization, service consumers are more likely to state their interest in, e. g., a fast response, the overall costs, or the used energy in general, ignoring the individual parameter categories. In order to support this approach, the overarching weight will be used for all individual parameters unless more specific weights are provided. For instance, if only the weight α for the Time is specified, the ExecutionTime, StorageTime, MemoryTime, and NetworkTime will all be scaled by this weight. If an individual weight is specified, e.g. α_4 for the NetworkTime, the remaining parameter categories still use the overarching α, but the NetworkTime will use α_4 instead.

Table 5.2: This table shows the mapping of the weights to the individual parameters including the parameter categories as introduced in Figure 5.5 with the first and third column specifying the parameters and the second and fourth column showing the corresponding weighted version.

Parameter	Weighted Parameter	Parameter	Weighted Parameter
Time (T)	α**T**	**Cost (C)**	β**C**
ExecutionTime (ET)	α_1ET	ExecutionCost (EC)	β_1EC
StorageTime (ST)	α_2ST	StorageCost (SC)	β_2SC
MemoryTime (MT)	α_3MT	MemoryCost (MC)	β_3MC
NetworkTime (NT)	α_4NT	NetworkCost (NC)	β_4NC
		ServiceCost (SeC)	β_5SeC
Energy (E)	γ**E**	**Sustainability (S)**	δ**S**
ExecutionEnergy (EE)	γ_1EE	ExecutionSustainability (ES)	δ_1ES
StorageEnergy (SE)	γ_2SE	StorageSustainability (SS)	δ_2SS
MemoryEnergy (ME)	γ_3ME	MemorySustainability (MS)	δ_3MS
NetworkEnergy (NE)	γ_4NE	NetworkSustainability (NS)	δ_4NS
DataAmount (DA)	ϵ**DA**	**Availability (A)**	ζ**A**
ExecutionDataAmount (EDA)	ϵ_1EDA	ExecutionAvailability (EA)	ζ_1EA
StorageDataAmount (SDA)	ϵ_2SDA	StorageAvailability (SA)	ζ_2SA
MemoryDataAmount (MDA)	ϵ_3MDA	MemoryAvailability (MA)	ζ_3MA
NetworkDataAmount (NDA)	ϵ_4NDA	NetworkAvailability (NA)	ζ_4NA
Reliability (R)	η**R**	**Maintainability (M)**	θ**M**
ExecutionReliability (ER)	η_1ER	ExecutionMaintainability (EM)	θ_1EM
StorageReliability (SR)	η_2SR	StorageMaintainability (SM)	θ_3SM
MemoryReliability (MR)	η_3MR	MemoryMaintainability (MM)	θ_2MM
NetworkReliability (NR)	η_4NR	NetworkMaintainability (NM)	θ_4NM
Bandwidth (B)	ι**B**	**Affiliation (Aff)**	κ**Aff**
History (H)	λ**H**	**Locality (L)**	μ**L**
Extensibility (Ex)	ν**Ex**	**Fidelity (F)**	ξ**F**
Usability (U)	π**U**	**Documentability (D)**	ρ**D**

In addition to the weights, service consumers might have a certain maximum or minimum value which a parameter should not exceed. We call these values *Limits* as addressed in Chapter 2. Each limit can provide one of three different limits as described in Equation 9. We differentiate between three types of limits: maximum value limits, minimum value limits, and minimum & maximum value limits.

Equation 9: Limits

$X \in [0, L_{max}]$

$X \in [L_{min}, 0]$

$X \in [L_{min}, L_{max}]$

With X being the value of the given parameter and L_{min}, L_{max} describing the given limits.

For example, these limits are used if a car manufacturer wants to use a service but only for a maximum price or in a certain timeframe specifying a maximum time before which a service has to be executed. Both, weights and limits can be used solely on the service consumers side, creating their own rankings of service providers or even ignore service providers not meeting their limits. On the other hand, those priorities and limits might be interesting for service providers to adjust their services if possible enabling service negotiations.

5.2 Quality of Service Vector

This section addresses the specifications and calculations for the quality of service vector based on the parameters introduced in the previous section. After the definition of the QoS vector itself, we address the five activities that a service consumer performs to create an ordered service provider list, as shown in the beginning of this chapter in Figure 5.3, and finally uses to select a service provider that fits the best to their need. In the end, we provide an example for the entire process.

5.2.1 Definition

In order to further evaluate the presented quality of service parameters which are spread across different levels of complexity, we use the representation of a quality of service vector (QoS VECTOR). This approach has been used several times for different tasks and approaches as discussed in Section 2.1.3. Equation 10 shows the entire QoS vector as defined by the quality of service parameters introduced in Section 5.1.

In the remaining of this chapter, we limit the QoS vector parameters to *Time*, *Cost*, *Energy*, *Data Amount*, *History*, *Bandwidth*, and *Locality* to explain the remaining concepts in a compact way. These parameters are representatives for the entire set of parameters due to their dependencies and levels. As *Data Amount* has different categories, as described in Section 5.1.2, we will consider it as a single parameter due to the fact that all of its categories have the same dependency; as this is not the case for *Time*, *Cost*, and *Energy*, we will list them separately. Nevertheless, all concepts also apply to the entire QoS vector with all its parameters.

Equation 10: Quality of Service Vector

$$\overrightarrow{QoS} := \begin{pmatrix} ExecutionTime \\ StorageTime \\ MemoryTime \\ NetworkTime \\ ExecutionCost \\ StorageCost \\ MemoryCost \\ NetworkCost \\ ServiceCost \\ ExecutionEnergy \\ StorageEnergy \\ MemoryEnergy \\ NetworkEnergy \\ ExecutionSustainability \\ StorageSustainability \\ MemorySustainability \\ NetworkSustainability \\ ExecutionDataAmount \\ StorageDataAmount \\ MemoryDataAmount \\ NetworkDataAmount \\ ExecutionAvailability \\ StorageAvailability \\ MemoryAvailability \\ NetworkAvailability \\ ExecutionReliability \\ StorageReliability \\ MemoryReliability \\ NetworkReliability \\ ExecutionMaintainability \\ StorageMaintainability \\ MemoryMaintainability \\ NetworkMaintainability \\ Bandwidth \\ Affiliation \\ History \\ Locality \\ Extensibility \\ Fidelity \\ Usability \\ Documentability \end{pmatrix} = \begin{pmatrix} ET \\ ST \\ MT \\ NT \\ EC \\ SC \\ MC \\ NC \\ SeC \\ EE \\ SE \\ ME \\ NE \\ ES \\ SS \\ MS \\ NS \\ EDA \\ SDA \\ MDA \\ NDA \\ EA \\ SA \\ MA \\ NDA \\ ER \\ SR \\ MR \\ NR \\ EM \\ SM \\ MM \\ NM \\ B \\ Aff \\ H \\ L \\ Ex \\ F \\ U \\ D \end{pmatrix}$$

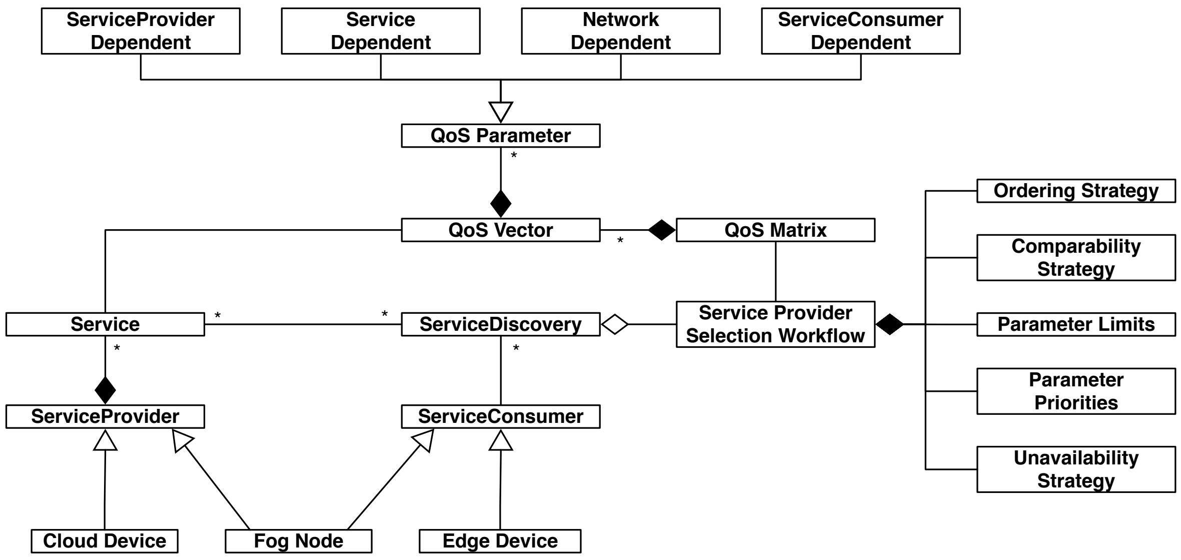


Figure 5.9: Overview of the QoS based xFogStar workflow. The workflow is used to select the best fitting `Service Provider` who offers a requested `Service` for the `Service Consumer`.

Figure 5.9 provides a preview of the relations between the different concepts that are introduced in this chapter.

As introduced in the beginning of this chapter, we mainly focus on the Fog Components that relate to the `Service Consumer` and `Service Provider`. Based on the layer definitions provided in Section 4.1.2, `Service Providers` can either be `Cloud Devices` or `Fog Nodes`. `Service Consumers` on the other hand are `Fog Nodes` or `Edge Devices`. Each `Service Provider` offers and advertises `Services` which can be discovered by the `Service Consumers`. To select the best fitting `Service Provider`, the `Service Consumer` uses the `Service Provider Selection Workflow` which includes the five parts: `Ordering Strategy`, `Comparability Strategy`, we `Parameter Limits`, `Parameter Priorities`, and `Unavailability Strategy`. We use a `QoS Vector` to specify the properties of the `Service` as well as the `Service Provider` and match them with the needs of the `Service Consumer`. The different `QoS Vectors` are gathered within a `QoS Matrix`. Every `QoS Vector` consists of several `QoS Parameters` which have different dependencies as introduced in Section 5.1.2.

Definition 33 shows the mathematical description of the QoS vector as described in Equation 10 for a specified *Fog Component X* which serves as the service consumer requesting a service and the *Fog Component Y* which serves as the service provider advertising that service. It is read as "The quality of service vector for a service requested by X and offered & advertised by Y". While Chapter 3-Chapter 4 highly focus on the differentiation between Fog Component types, in this chapter we focus on the Fog Component's roles as service consumers and providers while the previous concepts still apply.

Definition 33: QoS(X,Y)

$\overrightarrow{QoS}(X,Y)$ is defined as the quality of service vector of a service requested by a Fog Component X provided by a Fog Component Y.

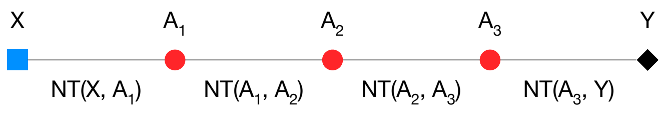
The written out version of the QoS Vector as shown in Definition 33, can be seen in Equation 11. It specifies how the limitation of the QoS Vector to Fog Components X and Y is transferred to the single parameters. As already indicated by the dependencies earlier, all parameters except the *History* and *Locality* are independent of the service consumer, and therefore do not need the X parameter. The *NetworkTime*, *NetworkCost*, *NetworkEnergy*, *Data Amount*, and *Bandwidth* are special in that sense that they also do not need to know about the service provider, and thus Y.

Equation 11: QoS(X,Y)

$$\overrightarrow{QoS}(X,Y) := \begin{pmatrix} ExecutionTime(X,Y) \\ StorageTime(X,Y) \\ MemoryTime(X,Y) \\ NetworkTime(X,Y) \\ ExecutionCost(X,Y) \\ StorageCost(X,Y) \\ MemoryCost(X,Y) \\ NetworkCost(X,Y) \\ ServiceCost(X,Y) \\ ExecutionEnergy(X,Y) \\ StorageEnergy(X,Y) \\ MemoryEnergy(X,Y) \\ NetworkEnergy(X,Y) \\ DataAmount(X,Y) \\ Bandwidth(X,Y) \\ History(X,Y) \\ Locality(X,Y) \end{pmatrix} = \begin{pmatrix} ET(X,Y) \\ ST(X,Y) \\ MT(X,Y) \\ NT(X,Y) \\ EC(X,Y) \\ SC(X,Y) \\ MC(X,Y) \\ NC(X,Y) \\ SeC(X,Y) \\ EE(X,Y) \\ SE(X,Y) \\ ME(X,Y) \\ NE(X,Y) \\ DA(X,Y) \\ B(X,Y) \\ H(X,Y) \\ L(X,Y) \end{pmatrix} = \begin{pmatrix} ET(Y) \\ ST(Y) \\ MT(Y) \\ NT \\ EC(Y) \\ SC(Y) \\ MC(Y) \\ NC \\ SeC(Y) \\ EE(Y) \\ SE(Y) \\ ME(Y) \\ NE \\ DA \\ B \\ H(X,Y) \\ L(X,Y) \end{pmatrix}$$

As shown in Section 3.4.1, the Fog Component X that requests a service from a Fog Component Y does not necessarily need to have the *Locality* of 0 or 1. If the Locality is greater 1, each parameter consists of several partitions. An example is shown in Equation 12 for the *Network Time* from service consumer X to service provider Y over 3 hops A_1, A_2, and A_3.

Equation 12: Locality Example



In the given example, the *Network Time* between Fog Component X and Fog Component Y is partitioned in four *Network Times*.

$$\mathrm{NT}(X, Y) = \mathrm{NT}(X, A_1) + \mathrm{NT}(A_1, A_2) + \mathrm{NT}(A_2, A_3) + \mathrm{NT}(A_3, Y)$$

Depending on the parameter, the definition how parameter are concatenated changes. While the *Network Time* is just added up, other parameters do not behave in the same way. Table 5.3 shows how the presented parameters can be connected among several hops from service consumer X (A_0) to service provider Y (A_{n+1}).

While the limitation of the QoS vector to a specific service consumer and specific provider reduces the complexity of some parameters, Definition 34 allows even further simplification by specifying the involved service s. It is read as "The quality of service vector for service S requested by X and offered & advertised by Y". Based on the previously introduced sets in Section 3.2.6, both Fog Components X and Y need to be within the set: $\mathrm{Interest}(S)$.

Definition 34: QoS$_S$(X,Y)

$\overrightarrow{QoS_S}(X, Y)$ is the quality of service vector for a service S requested by a Fog Component X provided by a Fog Component Y.

Table 5.3: This table shows how the listed parameters are calculated in dependency on the *Locality*, and therefore the amount of hops in between the service consumer and service provider. If *Locality* is 0, which means that the service consumer is its own provider, no calculation is needed. Otherwise, the calculations listed in *Hop Calculation* apply.

Parameter	**Short Form**	**Hop Calculation**
Time	$T(X, Y)$	$\sum_{i=0}^{n} T(A_i, A_i + 1)$
Cost	$C(X, Y)$	$\sum_{i=0}^{n} C(A_i, A_i + 1)$
Energy	$E(X, Y)$	$\sum_{i=0}^{n} E(A_i, A_i + 1)$
Sustainability	$S(X, Y)$	$\frac{\sum_{i=0}^{n} (E(A_i, A_{i+1}) \times S(A_i, A_{i+1}))}{\sum_{i=0}^{n} E(A_i, A_{i+1})}$
Data Amount	$DA(X, Y)$	$\sum_{i=0}^{n} DA(A_i, A_i + 1)$
Availability	$A(X, Y)$	$\prod_{i=0}^{n} A(A_i, A_i + 1)$
Reliability	$R(X, Y)$	$\sum_{i=0}^{n} R(A_i, A_i + 1)$
Maintainability	$M(X, Y)$	$\sum_{i=0}^{n} M(A_i, A_i + 1)$
Bandwidth	$B(X, Y)$	$MIN_{i=0}^{n} B(A_i, A_i + 1)$
Affiliation	$Aff(X, Y)$	$Aff(X, Y)$
History	$H(X, Y)$	$\prod_{i=0}^{n} H(A_i, A_i + 1)$
Locality	$L(X, Y)$	$\sum_{i=0}^{n} 1$
Extensibility	$Ex(X, Y)$	$MIN_{i=0}^{n} Ex(A_i, A_i + 1)$
Fidelity	$F(X, Y)$	$MIN_{i=0}^{n} F(A_i, A_i + 1)$
Usability	$U(X, Y)$	$MIN_{i=0}^{n} U(A_i, A_i + 1)$
Documentability	$D(X, Y)$	$MIN_{i=0}^{n} D(A_i, A_i + 1)$

Equation 13 shows the written out form including the service limitation. As before, parameters independent of the service are shown without the service specification. Therefore, parameters without the specification of service S are the *History*, *Bandwidth*, and *Locality*.

Equation 13: QoS$_S$(X,Y)

$$\overrightarrow{QoS_S}(X,Y) := \begin{pmatrix} ExecutionTime_S(Y) \\ StorageTime_S(Y) \\ MemoryTime_S(Y) \\ NetworkTime_S \\ ExecutionCost_S(Y) \\ StorageCost_S(Y) \\ MemoryCost_S(Y) \\ NetworkCost_S \\ ServiceCost_S(Y) \\ ExecutionEnergy_S(Y) \\ StorageEnergy_S(Y) \\ MemoryEnergy_S(Y) \\ NetworkEnergy_S \\ DataAmount_S \\ Bandwidth_S \\ History_S(X,Y) \\ Locality_S(X,Y) \end{pmatrix} = \begin{pmatrix} ET_S(Y) \\ ST_S(Y) \\ MT_S(Y) \\ NT_S \\ EC_S(Y) \\ SC_S(Y) \\ MC_S(Y) \\ NC_S \\ SeC_S(Y) \\ EE_S(Y) \\ SE_S(Y) \\ ME_S(Y) \\ NE_S \\ DA_S \\ B_S \\ H_S(X,Y) \\ L_S(X,Y) \end{pmatrix} = \begin{pmatrix} ET_S(Y) \\ ST_S(Y) \\ MT_S(Y) \\ NT_S \\ EC_S(Y) \\ SC_S(Y) \\ MC_S(Y) \\ NC_S \\ SeC_S(Y) \\ EE_S(Y) \\ SE_S(Y) \\ ME_S(Y) \\ NE_S \\ DA_S \\ B \\ H(X,Y) \\ L(X,Y) \end{pmatrix}$$

As the final step, we allow each parameter to be weighted. Based on Table 5.2 and Equation 8, each parameter receives a unique weight in the range of $[0,1] \in \mathbb{Q}$ indicating the importance of the parameter as a percentage value. Using the Hadamard Product, as shown in Definition 35, we can element-wise combine vectors including all weights ($\overrightarrow{W}$) and the QoS vector ($\overrightarrow{QoS}$). With one weight per QoS vector entry, both vectors have the same dimensions. The resulting vector is of the same dimension as well, multiplying the first value of $\overrightarrow{W}$ with the first value of $\overrightarrow{QoS}$, then the second values of each vector and so on.

Definition 35: Quality of Service Prioritization

Using the Hadamard Product [69], individual weights $\overrightarrow{W}$ can be assigned for each QoS vector component allowing prioritization.

The written out weighted quality of service vector including the Fog Component limitation and the service limitation is shown in Equation 14 and represents the final definition of the QoS vector described within this dissertation.

Equation 14: Weighted, service restricted QoS

$\overrightarrow{QoS_S^W(X,Y)}$ is the quality of service vector for service S requested by X provided by Y $\overrightarrow{QoS_S(X,Y)}$ prioritized by weights $\overrightarrow{W}$.

$$\overrightarrow{QoS_S^W(X,Y)} := \overrightarrow{W} \circ \overrightarrow{QoS_S(X,Y)} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ \epsilon \\ \iota \\ \lambda \\ \mu \end{pmatrix} \circ \begin{pmatrix} ET_S(Y) \\ ST_S(Y) \\ MT_S(Y) \\ NT_S \\ EC_S(Y) \\ SC_S(Y) \\ MC_S(Y) \\ NC_S \\ SeC_S(Y) \\ EE_S(Y) \\ SE_S(Y) \\ ME_S(Y) \\ NE_S \\ DA_S \\ B \\ H(X,Y) \\ L(X,Y) \end{pmatrix} = \begin{pmatrix} \alpha_1 ET_S(Y) \\ \alpha_2 ST_S(Y) \\ \alpha_3 MT_S(Y) \\ \alpha_4 NT_S \\ \beta_1 EC_S(Y) \\ \beta_2 SC_S(Y) \\ \beta_3 MC_S(Y) \\ \beta_4 NC_S \\ \beta_5 SeC_S(Y) \\ \gamma_1 EE_S(Y) \\ \gamma_2 SE_S(Y) \\ \gamma_3 ME_S(Y) \\ \gamma_4 NE_S \\ \epsilon DA_S \\ \iota B \\ \lambda H(X,Y) \\ \mu L(X,Y) \end{pmatrix}$$

5.2.2 Comparability

While the previous sections described **which** parameters we use for comparing service providers offering a specific service, in the following sections we focus on **how** the presented QoS parameters and the resulting quality of service vector can be compared to create a ranking of which service provider is the best choice.

Equation 15 shows two QoS vectors with concrete values for each parameter that we use as an example throughout this section. For better readability, the parameter names and the according units are listed in between. The shown vectors describe two service providers Y_1 and Y_2 offering the same service S that have two hops between the service provider and the service consumer indicated by a Locality of 3. The previous experience of the service consumer with those service providers is also the same (75%).

Equation 15: QoS Vector Comparison

$$\overrightarrow{QoS_S(X,Y_1)} = \begin{pmatrix} 4 \\ 10 \\ 4 \\ 89 \\ 0.6 \\ 1.3 \\ 0.2 \\ 1.8 \\ 4.0 \\ 15.0 \\ 5.0 \\ 4.0 \\ 12.0 \\ 220 \\ 195 \\ 75 \\ 3 \end{pmatrix} \begin{array}{c} ExecutionTime\ (ms) \\ StorageTime\ (ms) \\ MemoryTime\ (ms) \\ NetworkTime\ (ms) \\ ExecutionCost\ (\$) \\ StorageCost\ (\$) \\ MemoryCost\ (\$) \\ NetworkCost\ (\$) \\ ServiceCost\ (\$) \\ ExecutionEnergy\ (W) \\ StorageEnergy\ (W) \\ MemoryEnergy\ (W) \\ NetworkEnergy\ (W) \\ DataAmount\ (B) \\ Bandwidth\ (MBit/s) \\ History\ (\%) \\ Locality \end{array} \begin{pmatrix} 5 \\ 9 \\ 3 \\ 60 \\ 0.5 \\ 1.2 \\ 0.3 \\ 2.0 \\ 5.0 \\ 17.0 \\ 4.0 \\ 3.0 \\ 14.0 \\ 200 \\ 200 \\ 75 \\ 3 \end{pmatrix} = \overrightarrow{QoS_S(X,Y_2)}$$

This raises the following questions:

1. How can we compare parameters of different vectors?
2. What is better for each parameter, higher values or lower values?
3. How can we handle not provided parameters?
4. How can we assign each QoS vector a value to create a ranking?
5. How can the service consumer influence the ranking of service providers?

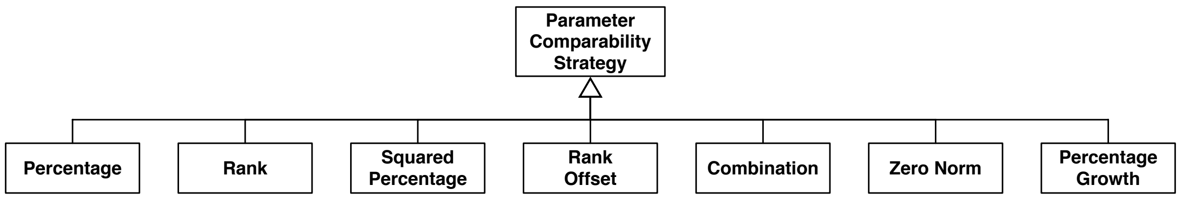


Figure 5.10: Hierarchy of different *Parameter Comparability Strategies* which can be used to compare individual parameters of quality of service vectors from different service providers.

Parameter - Comparability

Comparing different parameters of the same QoS vector is a difficult task. For instance, how much more money is it worth to save a couple of milliseconds time; or how much additional energy is acceptable for a higher transfer rate? In general these questions can only be answered based on the application domain and the application itself, as the values not only differ in their unit but also in their range of values. As for example, NetworkTime and MemoryTime are both measured in milliseconds, but MemoryTime usually gets dwarfed by the NetworkTime. Therefore, if we try to create one value to represent a QoS vector by adding up those parameters, the MemoryTime would be almost irrelevant due to the much bigger value of NetworkTime. This issue only gets worse when comparing parameters which do not even have the same unit.

We need a strategy that addresses the following attributes when comparing parameters: all parameters should have the same unit, and therefore be directly comparable, their values should be in the same range, and, in best case, the strategy should be resistant against outliers.

In the following, we present strategies that allow comparability between parameters by looking at several QoS vectors at once, reflecting a real-life scenario in which several service providers exist that offer the same service. These strategies are simple example strategies to show the occurring issues. Figure 5.10 presents a UML class diagram providing an overview of the strategies.

Equation 16 shows our example from Equation 15 in which the first two columns represent $QoS_S(X, Y_1)$ and $QoS_S(X, Y_2)$, but five additional service providers with their QoS vectors are introduced. Each row shows all values of the different QoS vectors for the same parameter, resulting in a matrix of dimension $m \times n$ – with n representing the amount of service providers and m representing the amount of parameters that are used to compare the vectors.

Equation 16: QoS Vector Matrix Example

$$\begin{array}{c} \text{ExecutionTime} \\ \text{StorageTime} \\ \text{MemoryTime} \\ \text{NetworkTime} \\ \text{ExecutionCost} \\ \text{StorageCost} \\ \text{MemoryCost} \\ \text{NetworkCost} \\ \text{ServiceCost} \\ \text{ExecutionEnergy} \\ \text{StorageEnergy} \\ \text{MemoryEnergy} \\ \text{NetworkEnergy} \\ \text{DataAmount} \\ \text{Bandwidth} \\ \text{History} \\ \text{Locality} \end{array} \begin{pmatrix} 4 & 5 & 9 & 2 & 8 & 4 & 7 \\ 10 & 9 & 8 & 5 & 14 & 13 & 28 \\ 4 & 3 & 1 & 5 & 4 & 7 & 4 \\ 89 & 60 & 70 & 75 & 95 & 111 & 45 \\ 0.6 & 0.5 & 0.3 & 0.7 & 0.9 & 1.2 & 1.8 \\ 1.3 & 1.2 & 1.5 & 1.7 & 2.1 & 0.8 & 1.0 \\ 0.2 & 0.3 & 0.6 & 0.1 & 0.2 & 0.3 & 0.6 \\ 1.8 & 2.0 & 2.0 & 3.0 & 2.4 & 2.1 & 5.1 \\ 4.0 & 5.0 & 6.6 & 5.4 & 7.3 & 9.1 & 7.8 \\ 15.0 & 17.0 & 12.0 & 19.0 & 21.0 & 27.0 & 36.0 \\ 5.0 & 4.0 & 3.0 & 7.0 & 8.4 & 7.4 & 9.1 \\ 4.0 & 3.0 & 2.0 & 7.3 & 2.4 & 7.9 & 1.3 \\ 12.0 & 14.0 & 13.2 & 14.3 & 12.2 & 12.8 & 17.2 \\ 220 & 200 & 190 & 178 & 198 & 232 & 243 \\ 195 & 200 & 32 & 64 & 128 & 100 & 120 \\ 75 & 75 & 90 & 78 & 85 & 60 & 67 \\ 3 & 3 & 7 & 4 & 3 & 4 & 2 \end{pmatrix}$$

The first strategy, called *Percentage*, creates relative values for each parameter by comparing each value to all other values of that parameter from different QoS vectors. Equation 17 shows the transposed *Network Time* row of the previous example. Each value is divided by the summarized value of all values for this parameter calculating the fraction of each value based on the entirety of *Network Time*. The resulting percentage values can be used to calculate how the different *Network Times* behave in comparison to each other as shown in Equation 18.

Equation 17: QoS Vector Matrix Row

$$\text{NetworkTime} \begin{pmatrix} 89 \\ 60 \\ 70 \\ 75 \\ 95 \\ 111 \\ 45 \end{pmatrix}^{T} \longrightarrow \begin{pmatrix} 89 \div 545 \\ 60 \div 545 \\ 70 \div 545 \\ 75 \div 545 \\ 95 \div 545 \\ 111 \div 545 \\ 45 \div 545 \end{pmatrix}^{T} = \begin{pmatrix} 0.163 \\ 0.110 \\ 0.128 \\ 0.138 \\ 0.174 \\ 0.204 \\ 0.082 \end{pmatrix}^{T} = \begin{pmatrix} 16.3\% \\ 11.0\% \\ 12.8\% \\ 13.8\% \\ 17.4\% \\ 20.4\% \\ 8.2\% \end{pmatrix}^{T}$$

Using relative percentage values for each parameter has the benefit of all parameters having the same unit and the same scale, making them comparable; however, outliers would still lead to extreme percentage values, and therefore reduce the comparability of smaller values. For example, if the *Network Time* of one service provider is 3 seconds and the 2 other service providers have *Network Times* of 20ms and 30ms, the percentage values would be 98.3%, 0.66% and 0.98%. This would indicate that service providers two and three do not differ that much, also the second value is 50% bigger than the first value.

Equation 18: QoS Vector Matrix Row Comparability

	16.3%	11.0%	12.8%	13.8%	17.4%	20.4%	8.2%
16.3%	± 0%	-32.5%	-21.5%	-15.3%	6.7%	25.2%	-49.7%
11.0%	48.2%	± 0%	16.4%	25.5%	58.2%	85.5%	-25.5%
12.8%	27.3%	-14.1%	± 0%	7.8%	35.9%	59.4%	35.9%
13.8%	18.1%	-20.3%	-7.2%	± 0%	26.1%	47.8%	-40.6%
17.4%	-6.3%	-36.8%	-26.4%	-20.7%	± 0%	17.2%	-52.9%
20.4%	-20.1%	-46.1%	-37.3%	-32.4%	-14.7%	± 0%	-59.8%
8.2%	98.8%	34.1%	56.1%	68.3%	112.2%	148.8%	± 0%

The second strategy, called *Rank*, focuses less on the size of the parameters, but on their order from minimum value to maximum value. The lowest value gets assigned a value of 0, the next bigger value a 1 and so on. Definition 36 shows the definition of the *Rank* function.

Definition 36: Rank

$\mathrm{Rank}(X, (Y_i)_{i=1,\dots,n}) :=$ {The position of X in $(Y_i)_{i=1,\dots,n}$ if $(Y_i)_{i=1,\dots,n}$ is sorted in descending order. }

with:

$X \in (Y_i)_{i=1,\dots,n})$

$\min(\mathrm{Rank}) = 0$

If identical values occur, they are assigned the same value, but the next value is increased by 1.

The main advantage of this strategy is the same unit and scale and the resistance of against outliers. While this is a huge benefit, the lack of comparison between the values might be an issue, as shown in Equation 19. If the *Network Time* of the first service provider is 20ms, but the *Network Time* of the another service provider is 3s, the rank values are close together, obfuscating the notable gap between them.

Equation 19: Rank Example

$$\text{NetworkTime} \begin{pmatrix} 20 \\ 3000 \\ 30 \\ 30 \\ 40 \end{pmatrix}^{\mathsf{T}} \longrightarrow \begin{pmatrix} 0 \\ 4 \\ 1 \\ 1 \\ 3 \end{pmatrix}^{\mathsf{T}}$$

The *Squard Percentage Strategy* squares the initial values before calculating the percentage value. Although this is more receptive against outliers, it allows better distinguishability between values that are close together, as shown in Equation 20.

Equation 20: Squared Percentage Example

$$\text{NetworkTime} \begin{pmatrix} 20 \\ 22 \\ 25 \\ 23 \\ 30 \end{pmatrix}^{\mathsf{T}} \longrightarrow \begin{pmatrix} 400 \\ 484 \\ 625 \\ 529 \\ 900 \end{pmatrix}^{\mathsf{T}} \longrightarrow \begin{pmatrix} 400 \div 2938 \\ 484 \div 2938 \\ 625 \div 2938 \\ 529 \div 2938 \\ 900 \div 2938 \end{pmatrix}^{\mathsf{T}} = \begin{pmatrix} 13.6\% \\ 16.5\% \\ 21.3\% \\ 18.0\% \\ 30.6\% \end{pmatrix}^{\mathsf{T}}$$

Another strategy, called *Rank Offset Strategy*, uses the Rank value but multiplies it with an offset to spread the values further apart. One approach for the offset is to equally spread the values between 0 and 100, making them comparable to percentage values. Thus, the offset is defined as $\frac{100}{N-1}$, where N is defined as the number of service providers. An example is shown in Equation 21.

Equation 21: Rank Offset Example

$$\text{NetworkTime} \begin{pmatrix} 20 \\ 22 \\ 25 \\ 23 \\ 30 \end{pmatrix}^{\mathsf{T}} \longrightarrow \begin{pmatrix} 0 \times \frac{100}{4} \\ 1 \times \frac{100}{4} \\ 3 \times \frac{100}{4} \\ 2 \times \frac{100}{4} \\ 4 \times \frac{100}{4} \end{pmatrix}^{\mathsf{T}} \longrightarrow \begin{pmatrix} 0 \\ 25 \\ 75 \\ 50 \\ 100 \end{pmatrix}^{\mathsf{T}}$$

Another strategy is the *Combination Strategy*, it combines the Rank Strategy with the Percentage Strategy. For this strategy, the resulting ranks and percentage values are multiplied by each other to create the new value. This new value has the benefit that all values have the same unit while maintaining the difference from the original values as shown in Equation 22.

Equation 22: Combination Strategy

$$\mathsf{NetworkTime}\begin{pmatrix}20\\22\\25\\23\\30\end{pmatrix}^{\mathsf{T}} \longrightarrow \begin{pmatrix}0\times 16.7\\1\times 18.3\\3\times 20.8\\2\times 19.2\\4\times 25.0\end{pmatrix}^{\mathsf{T}} \longrightarrow \begin{pmatrix}0\\18.3\\62.4\\38.4\\100\end{pmatrix}^{\mathsf{T}}$$

The *Zero Norm Strategy* uses the Percentage Strategy but sets the minimum value to zero. Therefore, the best value is highlighted even stronger while maintaining the benefits of the Percentage Strategy. An example is shown in Equation 23.

Equation 23: Zero Norm Example

$$\mathsf{NetworkTime}\begin{pmatrix}20\\22\\25\\23\\30\end{pmatrix}^{\mathsf{T}} \longrightarrow \begin{pmatrix}0\\22\\25\\23\\30\end{pmatrix}^{\mathsf{T}} \longrightarrow \begin{pmatrix}0\%\\22.0\%\\25.0\%\\23.0\%\\30.0\%\end{pmatrix}^{\mathsf{T}}$$

The last presented strategy is the *Percentage Growth Strategy*. It selects the minimum value as the value against which all other values are calculated in terms of growth. Therefore, the minimum value itself receives the percentage 0%. In comparison to the other strategies, this strategy allows direct comparisons between the values in the sense that a service consumer can see in how much increase the selection of a worse service provider would result. Based on the same example as before, Equation 24 shows the resulting values.

Equation 24: Percentage Growth

$$\mathsf{NetworkTime}\begin{pmatrix}20\\22\\25\\23\\30\end{pmatrix}^{\mathsf{T}} \longrightarrow \begin{pmatrix}0.0\%\\10.0\%\\25.0\%\\15.0\%\\50.0\%\end{pmatrix}^{\mathsf{T}}$$

Although we presented several strategies to create comparability between different values of the same parameter and also the parameters between each other, this list presents just the basic comparison strategies and is far from exclusive. Depending on the goal of the service consumer, many different strategies can be selected, comparing different aspects and highlighting others.

Parameter - Order

This paragraph addresses the second question that some parameters are considered "better" if they have bigger values and for some parameters lower values are preferred. While this does not create an issue per se, it needs to be addressed to create one value for the entirety of the parameters for each vector. Equation 25 provides an overview of all introduced parameters—excluding categories—with their unit and value range. While *Time*, *Cost*, *Energy*, *Data Amount*, *Maintainability*, and *Locality* should have low values, *Sustainability*, *Availability*, *Reliability*, *Bandwidth*, *Affiliation*, *History*, *Extensibility*, *Fidelity*, *Usability*, and *Documentability* should have high values.

Equation 25: Parameter Comparison

Time (ms): $T \in \mathbb{N}_0$
Cost ($): $C \in \mathbb{Q}_0^+$
Energy (W): $E \in \mathbb{Q}_0^+$
Sustainability (%): $S \in [0, 100]$
Data Amount (Bytes): $DA \in \mathbb{N}_0$
Availability (%): $A \in [0, 100]$
Reliability (ms): $R \in \mathbb{N}_0$
Maintainability (ms): $M \in \mathbb{N}_0$
Bandwidth (MBit/s): $B \in \mathbb{Q}_0^+$
Affiliation (Bool): $Aff \in \mathbb{B}$
History (%): $H \in [0, 100]$
Locality: $L \in \mathbb{N}_0$
Extensibility: $Ex \in \mathbb{N}$
Fidelity (%): $F \in [0, 100]$
Usability (%): $U \in [0, 100]$
Documentabiltiy (%): $D \in [0, 100]$

In order to unify the direction, either the first parameters need to be inverted to prefer higher values or the second parameters need to be adjusted to prefer low values. Several approaches can be used to achieve this—we will present three possibilities based on the concepts introduced in Section 5.2.2.

As inverting percentage values is simple by defining each percentage value p as $1 - p$, thus creating the complement event, we use this idea as the first approach.

Thus, we measure *Unsustainability*, *Unavailability*, redefine *History* to have lower values the better the previous experiences with the service provider were. We redefine *Fidelity* to measure the deviation of the provided service from its specification, redefine *Usability* as usability problems that occurred, and *Documentability* as the amount of missing documentation. While this approach works for percentage values, it is not possible for absolute values as no maximum value exists for all parameters from which we can subtract the current value. Using the comparability strategies from the previous section, and therefore transferring all parameters into percentage values, all parameters can be inverted to the opposite event.

Additionally, without creating percentage values, also absolute values can be inverted if we compare several QoS vectors at once. Thereby, we have several values per parameter and a maximum value among those. By subtracting all values from the maximum value, the ordering for the parameter can be inverted, switching previously considered ”good” values with ”bad” values. This works in both directions: If low values are preferred, but the best value is currently high or if high values are preferred, but the best value is currently low. Definition 37 shows the definition and an example is presented in Equation 26.

Definition 37: Parameter Ordering

Some parameters inherently prefer low values and some high values. To invert the scales, the following approach can be used:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \longrightarrow \begin{pmatrix} x_{max} - x_1 \\ x_{max} - x_2 \\ x_{max} - x_3 \\ x_{max} - x_4 \\ x_{max} - x_5 \\ x_{max} - x_6 \\ x_{max} - x_7 \end{pmatrix}$$

Equation 26: Parameter Ordering Example

$$\begin{pmatrix} 89 \\ 60 \\ 70 \\ 75 \\ 95 \\ 111 \\ 45 \end{pmatrix}^T \longrightarrow \begin{pmatrix} 22 \\ 51 \\ 41 \\ 36 \\ 16 \\ 0 \\ 66 \end{pmatrix}^T$$

Parameter - Unavailability

This paragraph addresses the third question that not all parameters might be available for all service providers. While service providers should always be able to provide Time, Cost, Bandwidth, History, and Locality, the other parameters might not be; creating a sparse matrix in comparison to the example provided in Equation 16. On the one side, service providers might not have the appropriate values available, and thus are not able to forward them; on the other side, if service providers have bad values for some parameters, they could intentionally omit those values. While the second aspect addresses a transparency issue, both aspects result in the same overarching issue: How can service consumers compare parameter values against missing values? When focusing on individual QoS vectors, this can easily be addressed by just removing that parameter entry, but when comparing different QoS vectors, those values potentially harm the selection process.

The first approach to address this issue is omitting all parameter for which not every service provider can provide values. While this approach works if just a few parameters are missing, the benefit of QoS vector discovery is reduced, if too many parameters are omitted resulting in a comparison of parameters that are independent of the service provider which can be collected by the service consumers themselves.

Another approach is the idea of providing default values for missing parameters. While this idea sounds reasonable at first, the selection of default values can have an impact on the selection of service providers, but also on the meaning of certain parameter values. If, for instance, the Availability is not provided, default values could, e. g., be 0 or 100. While 0 would prevent the service provider from ever being selected by a service consumer, a value of 100 would improve the chances to be selected, and therefore weaken the trust of service consumers in the discovery process.

The selection of default values is even harder if the parameter does not have usable maximum or minimum values, such as parameters that track time. Although the minimum value of 0 can be specified, the maximum value would be $+\infty$. A value of 0 would indicate zero delay, a value of $+\infty$ an infinite delay or in other words, a service that will never return a result. Another special case is the Reliability. A default value of 0 would indicate a constantly broken service provider and a value of $+\infty$ would be a perfect service provider that never faces any down times. On the one hand, a constantly broken service provider would never be selected which is harmful for the service provider itself, on the other side, no down times would give this service provider an unfair boost compared to others.

One alternative approach is the assignment of average values. Given the example from Equation 17, another service provider which could not provide a value for the *Network Time* would receive a value of 78ms. While this value is worse than the values of the "best" service providers, it is better than the "worse" service providers giving it a chance to be selected by service consumers. But this approach includes another issue. If service providers know that they have bad values for certain parameters, not providing the parameter would assign them a better value than they could report themselves. There should always be an incentive to report as many parameter values as possible and not an incentive for secrecy.

We define $Max+$ or $Min-$ which assign values to missing values that are slightly worse than the worst parameter value that was provided by another service provider during this workflow. Although this could result in bad values if there are some negative outliers within the service providers, a service provider hiding information will never be first choice. Depending on the order for the parameter in question, the value will be slightly higher than the maximum value of the other providers or slightly lower than the minimum value.

Equation 27 shows a comparison between the different approaches using the *Network Time* example introduced earlier.

Equation 27: Unavailability

$$\begin{array}{cccccc}
\textbf{Original} & \textbf{Omitted} & \textbf{Min-Default} & \textbf{Average} & \textbf{Min-} & \textbf{Max+} \\
(NetworkTime) & & & & & \\
\begin{pmatrix}89\\60\\70\\75\\-\\95\\111\\45\\-\end{pmatrix}^{T} &
\begin{pmatrix}89\\60\\70\\75\\95\\111\\45\end{pmatrix}^{T} &
\begin{pmatrix}89\\60\\70\\75\\0\\95\\111\\45\\0\end{pmatrix}^{T} &
\begin{pmatrix}89\\60\\70\\75\\78\\95\\111\\45\\78\end{pmatrix}^{T} &
\begin{pmatrix}89\\60\\70\\75\\40\\95\\111\\45\\40\end{pmatrix}^{T} &
\begin{pmatrix}89\\60\\70\\75\\120\\95\\111\\45\\120\end{pmatrix}^{T}
\end{array}$$

QoS Vector - Ranking

This paragraph focuses on the idea of creating one value for the entirety of all parameters within a QoS vector. We use this identifying value to create a ranking for all service providers to establish a sorted list indicating which service provider is considered the best fit down to the worst for the service consumer requesting a specific service. Also we do not present the value creation for each approach, all those concepts can and should be used in combination with the following concepts.

Definition 38 shows the value for a QoS vector to be defined as the l_1 **norm**, which means the sum of its absolute values. As all provided parameters are defined in the positive spectrum, also in different ranges and units, it removes the need for the modulus, making it the sum of all vector elements.

Definition 38: Quality of Service Vector Value

The value of the QoS vector is defined by its l_1 **norm**.

$||\overrightarrow{QoS}|| = \sum_{i=1}^{n} |x_i|$ with x_i being the single vector components.

$$\begin{aligned}||\overrightarrow{QoS}|| = {} & |ET| + |ST| + |MT| + |NT| + |EC| + |SC| + |MC| + |NC| + |SeC| + \\ & |EE| + |SE| + |ME| + |NE| + |ES| + |SS| + |MS| + |NS| + |EDA| + \\ & |SDA| + |MDA| + |NDA| + |EA| + |SA| + |MA| + |NA| + |ER| + \\ & |SR| + |MR| + |NR| + |EM| + |SM| + |MM| + |NM| + |B| + |Aff| + \\ & |H| + |L| + |Ex| + |F| + |U| + |D|\end{aligned}$$

With:

$$\begin{aligned}& ET, ST, MT, NT, EDA, SDA, MDA, NDA \in \mathbb{N}_0 \\ & ER, SR, MR, NR, EM, SM, MM, NM, L \in \mathbb{N}_0 \\ & EC, SC, MC, NC, SeC, EE, SE, ME, NE, B \in \mathbb{Q}_0^+ \\ & ES, SS, MS, NS, EA, SA, MA, NA, H, F, U, D \in [0, 100] \\ & Affiliation \in \mathbb{B} \\ & Ex \in \mathbb{N}\end{aligned}$$

$$\begin{aligned}||\overrightarrow{QoS}|| = {} & ET + ST + MT + NT + EC + SC + MC + NC + SeC + \\ & EE + SE + ME + NE + ES + SS + MS + NS + EDA + \\ & SDA + MDA + NDA + EA + SA + MA + NA + ER + \\ & SR + MR + NR + EM + SM + MM + NM + B + Aff + \\ & H + L + Ex + F + U + D\end{aligned}$$

Using this definition, it is obvious why the parameter comparability approaches are optional, but the parameter ordering (Equation 5.2.2) and the parameter unavailability (Equation 5.2.2) are not. Having the same order for all parameters also defines the order for the QoS vector value. If all parameters should be low, the QoS vector value also needs to be low to be considered good, if the parameters should be big, the same holds true for the QoS vector value.

The unavailability of some parameters would artificially lower the QoS vector value which can be good or bad for the corresponding service provider. The different concepts shown in Equation 27 to prevent unavailable parameters can solve this, but need to be chosen based on the application domain.

We combine the definitions as introduced in Section 5.1 with the QoS vector value (Definition 38) including the dependencies in the l_1 **norm**.

Corollary 1 uses Definition 33 in combination with Definition 38 to introduce the limitations to service consumer X and service provider Y into the QoS vector value calculation and Corollary 2 adds the service limitation using Definition 34.

Corollary 1: Quality of Service Vector Value

Using Definition 33 and Definition 38, the quality of service vector value for a Fog Component X requesting a service from a Fog Component Y is defined as:

$$\begin{aligned}||\overrightarrow{QoS(X,Y)}|| = {} & ET(Y) + ST(Y) + MT(Y) + NT + EC(Y) + SC(Y) + MC(Y) + \\ & NC + SeC(Y) + EE(Y) + SE(Y) + ME(Y) + NE + ES(Y) + \\ & SS(Y) + MS(Y) + NS + EDA + SDA + MDA + NDA + \\ & EA(Y) + SA(Y) + MA(Y) + NA + ER(Y) + SR(Y) + MR(Y) + \\ & NR + EM(Y) + SM(Y) + MM(Y) + NM + B + Aff(X) + \\ & H(X,Y) + L(X,Y) + Ex + F(X,Y) + U(X,Y) + D(Y)\end{aligned}$$

Corollary 2: Quality of Service Vector Value (2)

Using Corollary 1 and Definition 34, the quality of service vector value for a Fog Component X requesting a service S from a Fog Component Y is defined as:

$$\begin{aligned}||\overrightarrow{QoS(X,Y)_S}|| = {} & ET(Y) + ST(Y) + MT(Y) + NT_S + EC(Y) + SC(Y) + MC(Y) + \\ & NC + SeC(Y) + EE(Y) + SE(Y) + ME(Y) + NE_S + ES(Y) + \\ & SS(Y) + MS(Y) + NS + EDA_S + SDA_S + MDA_S + NDA_S + \\ & EA(Y) + SA(Y) + MA(Y) + NA + ER(Y) + SR(Y) + MR(Y) + \\ & NR + EM(Y) + SM(Y) + MM(Y) + NM + B + Aff(X) + \\ & H(X,Y) + L(X,Y) + Ex_S + F(X,Y) + U(X,Y) + D(Y)\end{aligned}$$

QoS Vector - Prioritization

In this final paragraph, we address how QoS vectors are prioritized and how service consumers can influence which service providers are their best choice. Therefore, Corollary 3 uses Definition 38 and Definition 35 to introduce weights into the QoS vector value calculation.

Corollary 3: Weighted Quality of Service Vector Value

Using Definition 35 and Definition 38, the quality of service vector value using weights W is defined as:

$$\begin{aligned}||\overrightarrow{QoS}^{W}|| = {} & \alpha_1 ET + \alpha_2 ST + \alpha_3 MT + \alpha_4 NT + \beta_1 EC + \beta_2 SC + \beta_3 MC + \beta_4 NC + \\ & \beta_5 SeC + \gamma_1 EE + \gamma_2 SE + \gamma_3 ME + \gamma_4 NE + \delta_1 ES + \delta_2 SS + \delta_3 MS + \\ & \delta_4 NS + \epsilon_1 EDA + \epsilon_2 SDA + \epsilon_3 MDA + \epsilon_4 NDA + \zeta_1 EA + \zeta_2 SA + \\ & \zeta_3 MA + \zeta_4 NA + \eta_1 ER + \eta_2 SR + \eta_3 MR + \eta_4 NR + \theta_1 EM + \theta_2 SM + \\ & \theta_3 MM + \theta_4 NM + \iota B + \kappa Aff + \lambda H + \mu L + \nu Ex + \xi F + \pi U + \rho D\end{aligned}$$

Finally, Corollary 4 combines Corollary 2 and Corollary 3 to create the final equation that is used to calculate the QoS vector value limited to Fog Components, the requested service, and weights.

Corollary 4: Weighted, service restricted Quality of Service Vector Value

Using Corollary 2 and Corollary 3, the quality of service vector value for X requesting service S from Y using weights W is defined as:

$$\begin{aligned}||\overrightarrow{QoS(X,Y)}_S^W|| = {} & \alpha_1 ET(Y) + \alpha_2 ST(Y) + \alpha_3 MT(Y) + \alpha_4 NT_S + \beta_1 EC(Y) + \\ & \beta_2 SC(Y) + \beta_3 MC(Y) + \beta_4 NC + \beta_5 SeC(Y) + \gamma_1 EE(Y) + \\ & \gamma_2 SE(Y) \gamma_3 ME(Y) + \gamma_4 NE_S + \delta_1 ES(Y) + \delta_2 SS(Y) + \\ & \delta_3 MS(Y) + \delta_4 NS + \epsilon_1 EDA_S + \epsilon_2 SDA_S + \epsilon_3 MDA_S + \\ & \epsilon_4 NDA_S + \zeta_1 EA(Y) + \zeta_2 SA(Y) + \zeta_3 MA(Y) + \zeta_4 NA + \\ & \eta_1 ER(Y) + \eta_2 SR(Y) + \eta_3 MR(Y) + \eta_4 NR + \theta_1 EM(Y) + \\ & \theta_2 SM(Y) + \theta_3 MM(Y) + \theta_4 NM + \iota B + \kappa Aff(X) + \\ & \lambda H(X,Y) + \mu L(X,Y) + \nu Ex_S + \xi F(X,Y) + \pi U(X,Y) + \\ & \rho D(Y)\end{aligned}$$

Using Corollary 4 allows service consumers to assign weights, as already introduced in Section 5.1.4, to specify which parameters are of which importance. Based on those weights, the QoS vector value calculation highly depends on the service consumer, and therefore the application domain for which the service is requested.

5.2.3 Instantiation

This section shows an example instantiation of the QoS discovery worklow for the ranking of the service providers based on their QoS vectors from initial values to the final ranking, as introduced in the dynamic model Figure 5.3. For consistency, we use the same QoS parameter selection as in the previous sections considering the different *Times*, *Costs*, *Energy* consumptions, *Data Amounts*, *Bandwidths*, *Histories*, and *Localities*. The example QoS matrix for a service consumer requesting a service is shown in Equation 28 and contains QoS vectors for seven service providers.

Equation 28: QoS Example

$$\begin{array}{c} \mathsf{ExecutionTime} \\ \mathsf{StorageTime} \\ \mathsf{MemoryTime} \\ \mathsf{NetworkTime} \\ \mathsf{ExecutionCost} \\ \mathsf{StorageCost} \\ \mathsf{MemoryCost} \\ \mathsf{NetworkCost} \\ \mathsf{ServiceCost} \\ \mathsf{ExecutionEnergy} \\ \mathsf{StorageEnergy} \\ \mathsf{MemoryEnergy} \\ \mathsf{NetworkEnergy} \\ \mathsf{DataAmount} \\ \mathsf{Bandwidth} \\ \mathsf{History} \\ \mathsf{Locality} \end{array} \begin{pmatrix} 4 & 5 & 9 & 2 & 8 & 4 & 7 \\ 10 & 9 & 8 & 5 & 14 & 13 & 28 \\ 4 & 3 & 1 & 5 & 4 & 7 & 4 \\ 89 & 60 & 70 & 75 & 95 & 111 & 45 \\ 0.6 & 0.5 & 0.3 & 0.7 & 0.9 & 1.2 & 1.8 \\ 1.3 & 1.2 & 1.5 & 1.7 & 2.1 & 0.8 & 1.0 \\ 0.2 & 0.3 & 0.6 & 0.1 & 0.2 & 0.3 & 0.6 \\ 1.8 & 2.0 & 2.0 & 3.0 & 2.4 & 2.1 & 5.1 \\ 4.0 & 5.0 & 6.6 & 5.4 & 7.3 & 9.1 & 7.8 \\ 15.0 & 17.0 & 12.0 & 19.0 & 21.0 & 27.0 & 36.0 \\ 5.0 & 4.0 & 3.0 & 7.0 & 8.4 & 7.4 & 9.1 \\ 4.0 & 3.0 & 2.0 & 7.3 & 2.4 & 7.9 & 1.3 \\ 12.0 & 14.0 & 13.2 & 14.3 & 12.2 & 12.8 & 17.2 \\ 220 & 200 & 190 & 178 & 198 & 232 & 243 \\ 195 & 200 & 32 & 64 & 128 & 100 & 120 \\ 75 & 75 & 90 & 78 & 85 & 60 & 67 \\ 3 & 3 & 7 & 4 & 3 & 4 & 2 \end{pmatrix}$$

As this example already shows a fully populated QoS vector matrix, we do not have to `Select [an] Unavailability Strategy` or `Apply [an] Unavailability Strategy`.

Therefore, the first step is to define the acceptable limits for the service consumer, which requests the service, for each parameter. We want to consider all service providers who can offer the requested service for a total cost of less than 13$ and a total time of maximum 100ms. These limits are shown in Equation 29.

Equation 29: Limit Example

$$\mathsf{Time} \in [0, 100]$$
$$\mathsf{Cost} \in [0, 13]$$

After applying those limits, *Service Provider 5* is dropped due to a high total time, *Service Provider 6* based on both, the total time and the total cost, and *Service Provider 7* is too expensive. Therefore, the initial seven service providers are reduced to four. This is shown in Equation 30 with the red font highlighting the values which were too high and the red background color indicating dropped service providers. The resulting `Limited QoS Vector Matrix` is presented on the right.

Equation 30: Limited Matrix Example

$$\begin{array}{c}\mathbf{T}\\ \mathrm{ET}\\ \mathrm{ST}\\ \mathrm{MT}\\ \mathrm{NT}\\ \mathbf{C}\\ \mathrm{EC}\\ \mathrm{SC}\\ \mathrm{MC}\\ \mathrm{NC}\\ \mathrm{SeC}\\ \mathrm{EE}\\ \mathrm{SE}\\ \mathrm{ME}\\ \mathrm{NE}\\ \mathrm{DA}\\ \mathrm{B}\\ \mathrm{H}\\ \mathrm{L}\end{array}
\begin{pmatrix}
\mathbf{107} & \mathbf{77} & \mathbf{88} & \mathbf{87} & \mathbf{121} & \mathbf{135} & \mathbf{84}\\
4 & 5 & 9 & 2 & 8 & 4 & 7\\
10 & 9 & 8 & 5 & 14 & 13 & 28\\
4 & 3 & 1 & 5 & 4 & 7 & 4\\
89 & 60 & 70 & 75 & 95 & 111 & 45\\
\mathbf{7.9} & \mathbf{9.0} & \mathbf{11.0} & \mathbf{10.9} & \mathbf{12.9} & \mathbf{13.5} & \mathbf{16.3}\\
0.6 & 0.5 & 0.3 & 0.7 & 0.9 & 1.2 & 1.8\\
1.3 & 1.2 & 1.5 & 1.7 & 2.1 & 0.8 & 1.0\\
0.2 & 0.3 & 0.6 & 0.1 & 0.2 & 0.3 & 0.6\\
1.8 & 2.0 & 2.0 & 3.0 & 2.4 & 2.1 & 5.1\\
4.0 & 5.0 & 6.6 & 5.4 & 7.3 & 9.1 & 7.8\\
15.0 & 17.0 & 12.0 & 19.0 & 21.0 & 27.0 & 36.0\\
5.0 & 4.0 & 3.0 & 7.0 & 8.4 & 7.4 & 9.1\\
4.0 & 3.0 & 2.0 & 7.3 & 2.4 & 7.9 & 1.3\\
12.0 & 14.0 & 13.2 & 14.3 & 12.2 & 12.8 & 17.2\\
220 & 200 & 190 & 178 & 198 & 232 & 243\\
195 & 200 & 32 & 64 & 128 & 100 & 120\\
75 & 75 & 90 & 78 & 85 & 60 & 67\\
3 & 3 & 7 & 4 & 3 & 4 & 2
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\mathbf{107} & \mathbf{77} & \mathbf{88} & \mathbf{87}\\
4 & 5 & 9 & 2\\
10 & 9 & 8 & 5\\
4 & 3 & 1 & 5\\
89 & 60 & 70 & 75\\
\mathbf{7.9} & \mathbf{9.0} & \mathbf{11.0} & \mathbf{10.9}\\
0.6 & 0.5 & 0.3 & 0.7\\
1.3 & 1.2 & 1.5 & 1.7\\
0.2 & 0.3 & 0.6 & 0.1\\
1.8 & 2.0 & 2.0 & 3.0\\
4.0 & 5.0 & 6.6 & 5.4\\
15.0 & 17.0 & 12.0 & 19.0\\
5.0 & 4.0 & 3.0 & 7.0\\
4.0 & 3.0 & 2.0 & 7.3\\
12.0 & 14.0 & 13.2 & 14.3\\
220 & 200 & 190 & 178\\
195 & 200 & 32 & 64\\
75 & 75 & 90 & 78\\
3 & 3 & 7 & 4
\end{pmatrix}$$

For the second step, we have to `Select [a] Comparability Strategy`. We choose the *Combination Strategy* to get the benefits of using ranks and the benefits of using percentage values as described in Section 5.2.2. The `App[lication of the] Comparability Strategy` is shown in Equation 31. Using one intermediate step which shows the percentage values and the corresponding ranks, the equation shows the `Comparable QoS Vector Matrix` on the right.

Equation 31: Comparable Matrix Example

$$\begin{array}{c} \mathrm{ET} \\ \mathrm{ST} \\ \mathrm{MT} \\ \mathrm{NT} \\ \mathrm{EC} \\ \mathrm{SC} \\ \mathrm{MC} \\ \mathrm{NC} \\ \mathrm{SeC} \\ \mathrm{EE} \\ \mathrm{SE} \\ \mathrm{ME} \\ \mathrm{NE} \\ \mathrm{DA} \\ \mathrm{B} \\ \mathrm{H} \\ \mathrm{L} \end{array} \begin{pmatrix} 0.20\times 1 & 0.25\times 2 & 0.45\times 3 & 0.1\times 0 \\ 0.31\times 3 & 0.28\times 2 & 0.25\times 1 & 0.16\times 0 \\ 0.31\times 2 & 0.23\times 1 & 0.08\times 0 & 0.38\times 3 \\ 0.30\times 3 & 0.20\times 0 & 0.24\times 1 & 0.26\times 2 \\ 0.29\times 2 & 0.24\times 1 & 0.14\times 0 & 0.33\times 3 \\ 0.23\times 1 & 0.21\times 0 & 0.26\times 2 & 0.30\times 3 \\ 0.17\times 1 & 0.25\times 2 & 0.5\times 3 & 0.08\times 0 \\ 0.20\times 0 & 0.23\times 1 & 0.23\times 1 & 0.34\times 3 \\ 0.19\times 0 & 0.24\times 1 & 0.31\times 3 & 0.26\times 2 \\ 0.24\times 1 & 0.27\times 2 & 0.19\times 0 & 0.30\times 3 \\ 0.26\times 2 & 0.21\times 1 & 0.16\times 0 & 0.37\times 3 \\ 0.25\times 2 & 0.18\times 1 & 0.12\times 0 & 0.45\times 3 \\ 0.22\times 0 & 0.26\times 2 & 0.25\times 1 & 0.27\times 3 \\ 0.28\times 3 & 0.25\times 2 & 0.24\times 1 & 0.23\times 0 \\ 0.40\times 2 & 0.41\times 3 & 0.07\times 0 & 0.13\times 1 \\ 0.24\times 0 & 0.24\times 0 & 0.28\times 3 & 0.25\times 2 \\ 0.18\times 0 & 0.18\times 0 & 0.41\times 3 & 0.24\times 2 \end{pmatrix} = \begin{pmatrix} 0.20 & 0.50 & 1.35 & 0.00 \\ 0.93 & 0.56 & 0.25 & 0.00 \\ 0.62 & 0.23 & 0.00 & 1.14 \\ 0.90 & 0.00 & 0.24 & 0.52 \\ 0.58 & 0.24 & 0.00 & 0.99 \\ 0.23 & 0.00 & 0.52 & 0.90 \\ 0.17 & 0.50 & 1.50 & 0.00 \\ 0.00 & 0.23 & 0.23 & 1.02 \\ 0.00 & 0.24 & 0.93 & 0.52 \\ 0.24 & 0.54 & 0.00 & 0.90 \\ 0.52 & 0.21 & 0.00 & 1.11 \\ 0.50 & 0.18 & 0.00 & 1.35 \\ 0.00 & 0.52 & 0.25 & 0.81 \\ 0.84 & 0.50 & 0.24 & 0.00 \\ 0.80 & 1.23 & 0.00 & 0.13 \\ 0.00 & 0.00 & 0.84 & 0.50 \\ 0.00 & 0.00 & 1.23 & 0.48 \end{pmatrix}$$

The *Bandwidth* and the *History* do not inherently prefer lower values, thus we have to `Select [an] Ordering Strategy`. As we applied the *Combination Strategy* for parameter comparison, the resulting values are not percentage values, but scaled with their rank. Thus, we take the highest absolute value and subtract each value from it to invert the ordering. Equation 32 highlights with a red background the values that need to be inverted. After showing an intermediate step, the `Ordered QoS Vector Matrix` is presented.

Equation 32: Ordered Matrix Example

$$\begin{array}{c}\begin{array}{c|cccc} ET & 0.20 & 0.50 & 1.35 & 0.00 \\ ST & 0.93 & 0.56 & 0.25 & 0.00 \\ MT & 0.62 & 0.23 & 0.00 & 1.14 \\ NT & 0.90 & 0.00 & 0.24 & 0.52 \\ EC & 0.58 & 0.24 & 0.00 & 0.99 \\ SC & 0.23 & 0.00 & 0.52 & 0.90 \\ MC & 0.17 & 0.50 & 1.50 & 0.00 \\ NC & 0.00 & 0.23 & 0.23 & 1.02 \\ SeC & 0.00 & 0.24 & 0.93 & 0.52 \\ EE & 0.24 & 0.54 & 0.00 & 0.90 \\ SE & 0.52 & 0.21 & 0.00 & 1.11 \\ ME & 0.50 & 0.18 & 0.00 & 1.35 \\ NE & 0.00 & 0.52 & 0.25 & 0.81 \\ DA & 0.84 & 0.50 & 0.24 & 0.00 \\ B & 0.80 & 1.23 & 0.00 & 0.13 \\ H & 0.00 & 0.00 & 0.84 & 0.50 \\ L & 0.00 & 0.00 & 1.23 & 0.48 \end{array} \longrightarrow \\ \begin{pmatrix} 0.20 & 0.50 & 1.35 & 0.00 \\ 0.93 & 0.56 & 0.25 & 0.00 \\ 0.62 & 0.23 & 0.00 & 1.14 \\ 0.90 & 0.00 & 0.24 & 0.52 \\ 0.58 & 0.24 & 0.00 & 0.99 \\ 0.23 & 0.00 & 0.52 & 0.90 \\ 0.17 & 0.50 & 1.50 & 0.00 \\ 0.00 & 0.23 & 0.23 & 1.02 \\ 0.00 & 0.24 & 0.93 & 0.52 \\ 0.24 & 0.54 & 0.00 & 0.90 \\ 0.52 & 0.21 & 0.00 & 1.11 \\ 0.50 & 0.18 & 0.00 & 1.35 \\ 0.00 & 0.52 & 0.25 & 0.81 \\ 0.84 & 0.50 & 0.24 & 0.00 \\ 1.23-0.80 & 1.23-1.23 & 1.23-0.00 & 1.23-0.13 \\ 0.84-0.00 & 0.84-0.00 & 0.84-0.84 & 0.84-0.50 \\ 0.00 & 0.00 & 1.23 & 0.48 \end{pmatrix} = \begin{pmatrix} 0.20 & 0.50 & 1.35 & 0.00 \\ 0.93 & 0.56 & 0.25 & 0.00 \\ 0.62 & 0.23 & 0.00 & 1.14 \\ 0.90 & 0.00 & 0.24 & 0.52 \\ 0.58 & 0.24 & 0.00 & 0.99 \\ 0.23 & 0.00 & 0.52 & 0.90 \\ 0.17 & 0.50 & 1.50 & 0.00 \\ 0.00 & 0.23 & 0.23 & 1.02 \\ 0.00 & 0.24 & 0.93 & 0.52 \\ 0.24 & 0.54 & 0.00 & 0.90 \\ 0.52 & 0.21 & 0.00 & 1.11 \\ 0.50 & 0.18 & 0.00 & 1.35 \\ 0.00 & 0.52 & 0.25 & 0.81 \\ 0.84 & 0.50 & 0.24 & 0.00 \\ 0.43 & 0.00 & 1.23 & 1.10 \\ 0.84 & 0.84 & 0.00 & 0.34 \\ 0.00 & 0.00 & 1.23 & 0.48 \end{pmatrix}\end{array}$$

Table 5.4: Weight Example for the Parameter Importance specified by the service consumer.

Parameter	Weight	Parameter	Weight
Time (T)	$\alpha = 0.50$	Cost (C)	$\beta = 0.50$
ExecutionTime (ET)	$\alpha_1 = 0.50$	ExecutionCost (EC)	$\beta_1 = 0.50$
StorageTime (ST)	$\alpha_2 = 0.50$	StorageCost (SC)	$\beta_2 = 0.50$
MemoryTime (MT)	$\alpha_3 = 0.50$	MemoryCost (MC)	$\beta_3 = 0.50$
NetworkTime (NT)	$\alpha_4 = 0.50$	NetworkCost (NC)	$\beta_4 = 0.50$
		ServiceCost (SeC)	$\beta_5 = 0.50$
Energy (E)	$\gamma = 1.00$	Data Amount (DA)	$\epsilon = 0.50$
ExecutionEnergy (EE)	$\gamma_1 = 1.00$	ExecutionDataAmount (EDA)	$\epsilon_1 = 0.50$
StorageEnergy (SE)	$\gamma_2 = 1.00$	StorageDataAmount (SDA)	$\epsilon_2 = 0.50$
MemoryEnergy (ME)	$\gamma_3 = 1.00$	MemoryDataAmount (MDA)	$\epsilon_3 = 0.50$
NetworkEnergy (NE)	$\gamma_4 = 1.00$	NetworkDataAmount (NDA)	$\epsilon_4 = 0.50$
Bandwidth (B)	$\iota = 0.50$	History (H)	$\lambda = 0.50$
Locality (L)	$\mu = 0.50$		

Before creating the final ordered service provider list, the service consumer has the chance to `Define [the] Parameter Importance` by specifying a weight for each parameter according to Table 5.2. In the selection of the service provider we want to prioritize saving energy, therefore, lowering the weights for all other parameters by 50% which makes the consumed energy twice as important. The according weight-parameter mapping is shown in Table 5.4.

Finally, we `Create [a] QoS Vector Ranking` by multiplying each parameter from the resulting `Ordered QoS Vector Matrix` in Equation 32 by the weights specified in Table 5.4 and add up all resulting values for each service provider. This results in an `Orderer Service Provider List` as shown in Equation 33. According to the selected strategy for ordering the parameters, higher or lower values are preferred for the service providers: lower values in our example. In the equation, the energy values, which use a weight of 1.00, are highlighted with a green background and the service providers are highlighted in the order green, yellow, orange, and red indicating their fit for the service consumer. Therefore, the first service provider is selected.

Equation 33: Ordered Service Provider List Example

$$\begin{array}{c} \\ \text{ET} \\ \text{ST} \\ \text{MT} \\ \text{NT} \\ \text{EC} \\ \text{SC} \\ \text{MC} \\ \text{NC} \\ \text{SeC} \\ \text{EE} \\ \text{SE} \\ \text{ME} \\ \text{NE} \\ \text{DA} \\ \text{B} \\ \text{H} \\ \text{L} \end{array} \begin{pmatrix} 0.10 & 0.25 & 0.67 & 0.00 \\ 0.47 & 0.28 & 0.13 & 0.00 \\ 0.31 & 0.12 & 0.00 & 0.57 \\ 0.45 & 0.00 & 0.12 & 0.26 \\ 0.29 & 0.12 & 0.00 & 0.50 \\ 0.12 & 0.00 & 0.26 & 0.45 \\ 0.09 & 0.25 & 0.75 & 0.00 \\ 0.00 & 0.12 & 0.12 & 0.51 \\ 0.00 & 0.12 & 0.46 & 0.26 \\ 0.24 & 0.54 & 0.00 & 0.90 \\ 0.52 & 0.21 & 0.00 & 1.11 \\ 0.50 & 0.18 & 0.00 & 1.35 \\ 0.00 & 0.52 & 0.25 & 0.81 \\ 0.42 & 0.25 & 0.12 & 0.00 \\ 0.40 & 0.62 & 0.00 & 0.07 \\ 0.00 & 0.00 & 0.42 & 0.25 \\ 0.00 & 0.00 & 0.62 & 0.24 \end{pmatrix} \longrightarrow \begin{array}{cccc} \text{QoS}_1 & \text{QoS}_2 & \text{QoS}_3 & \text{QoS}_4 \\ \end{array} \begin{pmatrix} 3.51 & 3.58 & 3.92 & 7.28 \end{pmatrix}$$

Chapter 6

Validation of xFog

"First, theories and models are always simplifications. If they were as complex as reality, they would not be useful. Indeed, the value of theories is to cut through idiosyncracies and unearth similarities across cases."

— NICOLAJ SIGGELKOW [136]

This chapter presents the validation of the xFog framework and xFogStar addressing *Knowledge Goal 2* and *Knowledge Goal 3*. The validation tries to justify if stakeholder goals would be met if the treatment is implemented in the problem domain's context. It investigates if the requirements for the treatment are addressed within a model of the problem domain. As the validation is part of the design cycle, and thus conducted in a laboratory setting, the implementation of the treatment in the problem domain is not of interest, yet. This results in the validation being independent of the stakeholders, which is the main difference between validation and evaluation. Therefore, different research approaches are used. For the validation, we use modeling, simulations, and testing [148].

The validation of xFog is separated into three aspects of the xFog framework provided by xFogCore, xFogPlus, and xFogStar. For each of those validations, we relied on the validation approaches modeling and simulation. First, we introduce the design of the different case studies in Section 6.1. We present the problem domain of the case study, the requirements, and which concepts of xFog or xFogStar are addressed. We selected cases in different domains to support domain-independent conclusions. In total, we addressed six different domains: *Smart Environments*, *Smart Cities*, *Health*, *Continuous Integration*, *Metrology*, and *Gaming*. These domains were used in eight case studies mapped to three validations.

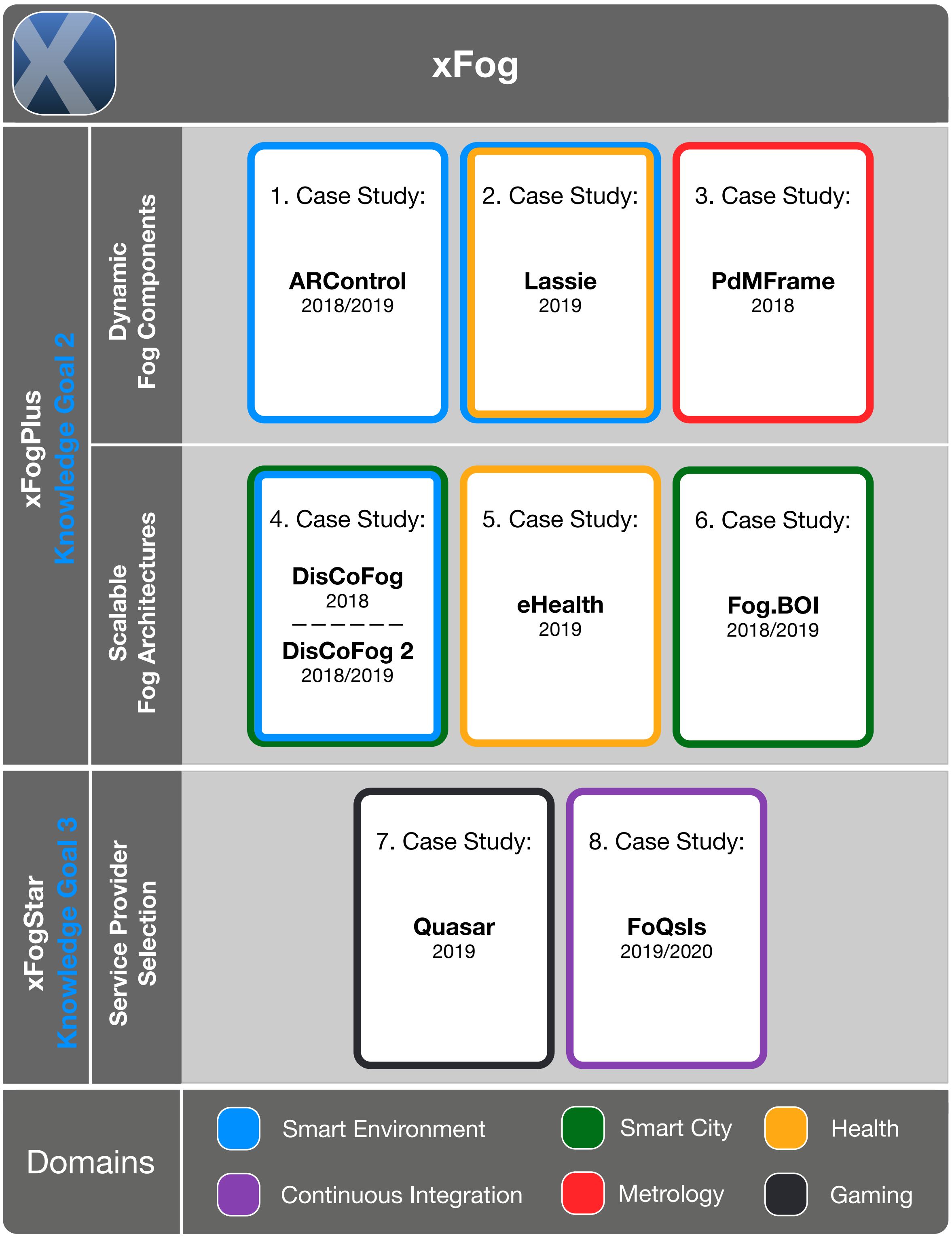

xFog
xFogPlus
Knowledge Goal 2
Dynamic Fog Components
1. Case Study: ARControl 2018/2019
2. Case Study: Lassie 2019
3. Case Study: PdMFrame 2018
Scalable Fog Architectures
4. Case Study: DisCoFog 2018
DisCoFog 2 2018/2019
5. Case Study: eHealth 2019
6. Case Study: Fog.BOI 2018/2019
xFogStar
Knowledge Goal 3
Service Provider Selection
7. Case Study: Quasar 2019
8. Case Study: FoQsIs 2019/2020
Domains
Smart Environment
Smart City
Health
Continuous Integration
Metrology
Gaming

Figure 6.1: An overview of the validation design for the xFog framework. Each of the three rows represents one validation with Dynamic Fog Components and Scalable Fog Architectures belonging to concepts related to xFogPlus and the Service Provider Selection belonging to the xFogStar workflow. The colored borders of the case studies represent the domains they belong to.

Second, in Section 6.2, we report on the results of the validation for the three core concepts: Dynamic Fog Components, Scalable Fog Architecture, and Service Provider Selection. While the formalization of Fog Computing (xFogCore) is used throughout all three concepts, each of the concepts can be assigned to an addition to xFog as shown in Figure 6.1. *Dynamic Fog Components* and *Scalable Fog Architectures* are covered in xFogPlus as shown in the first two rows of the diagram. The *Service Provider Selection* uses the core xFog concepts (xFogCore) and the workflow introduced by xFogStar.

Third, we discuss the impact of the results for the xFog framework in Section 6.3. We interpret the results and address threads to the validity of the validation.

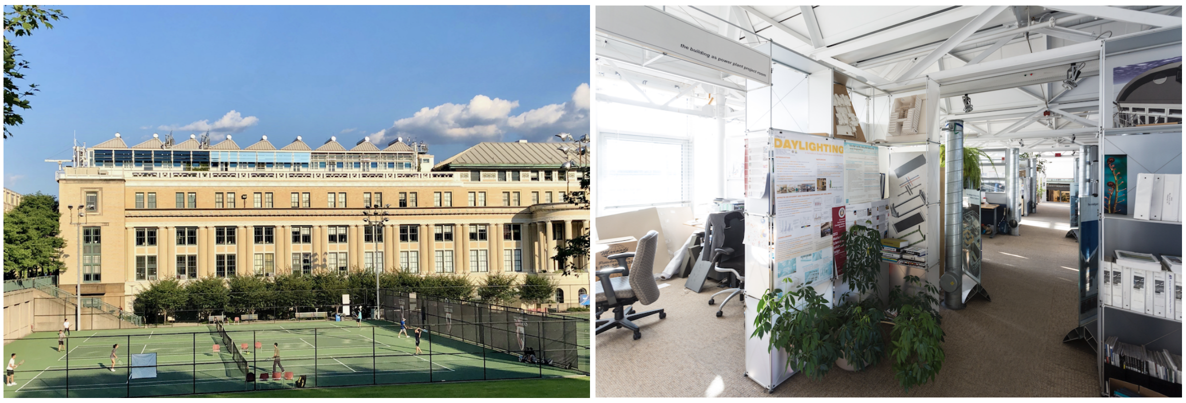
6.1 Case Study Design

We introduce the design of eight case studies that we conducted for the three validations. As shown in Figure 6.1, the first two validations are multiple case studies with three cases each and the third validation consists of two cases. Each case study within one validation is based in a different problem domain to support domain-independent conclusions. We outline the problem domains to establish the background of the developed systems as well as the stakeholders and their expectations. We describe the problem to be solved, the goals of each case study with the resulting requirements to be fulfilled, and the concepts of xFog and xFogStar that were used. Lastly, we explain the implementation process.

6.1.1 ARControl

The first case study, called *ARControl* is based within the domain of smart environments and is presented in "Fog Horizons–A Theoretical Concept to Enable Dynamic Fog Architectures" [66]. ARControl is short for "**A**ugmented **R**eality **Control**".

Context: As defined by Peters, smart environments are subclasses of instrumented spaces [110]: Instrumented spaces use actuators and sensors as well as rules to interact with users, while smart environments introduce service automation, and learns and adapts its behavior during use. This flexibility is supported by more complex reasoning capabilities. This domain was chosen based on its amount of different actuators and sensors, that need to be supported, the inherent hierarchical structure of buildings, and their dynamic surrounding with different occupants constantly entering and leaving the environment. Additionally, with the development of new actuators and sensors, new devices can be added to the smart environment as well as old devices maintained and replaced. The target environment for this case study was the Robert L. Preger Intelligent Workplace at Carnegie Mellon University [63].



(a) Outdoor (b) Indoor

Figure 6.2: The Intelligent Workplace at Carnegie Mellon University in Pittsburgh, PA, USA [63].

This lived-in laboratory is home to the researchers of the Center for Building Performance and Diagnostics. It was designed to significantly reduce its energy requirements compared to conventional office environments up to being completely self-sufficient. Thus, it incorporates a variety of devices, e. g., solar panels on its roof, ample natural lighting, natural ventilation with controllable windows, water-based cooling/heating systems, and occupancy sensors. Figure 6.2 gives an impression of the buildings exterior (a) and interior (b).

Stakeholders: Accordingly, two different classes of stakeholders were present: those that permanently work in this environment, e. g., the researchers or scientific staff, and those that temporary join, such as students of the Intelligent Workplace joining for lectures or student projects. Both classes were occupants in the smart environment, but with different goals and requirements. The possibilities of involvement also differ substantially: Permanent occupants could configure their environment to their needs, change the environment's setup and provide personal data to help the system learn their preferences. Temporary occupants, on the other hand, were usually not allowed to change the environment or provide data to support the adjustments to their needs for long time usage. Additionally, the knowledge of their surrounding smart environment differed between temporary and permanent occupants. Permanent occupants get used to the names of the devices which they use regularly, learn more about the environment from other co-workers, and are willing to adjust their workplace for personal comfort. As the Intelligent Workplace does not have simple light switches, knowing the names of the devices is crucial to control them via smartphone apps. Students joining the Intelligent Workplace do not know the devices name, and therefore, for example, struggle to turn on a light they want to use.

Goal: The goal of this case study was to address this difference in knowledge and involvement as well as to provide an intuitive concept to interact with the smart environment. This included controlling actuators and accessing data of sensors. With the emphasis of the Intelligent Workplace set on reducing energy consumption, the second goal was to visualize energy consumption of occupants to support energy awareness. These goals should be achieved using the hierarchical structure of smart environments and the locality provided by Fog Computing. The open space office of the Intelligent Workplace was structured as follows: Each workplace of the scientific staff had its own devices, e. g., computer, lights, and power plugs. These individual workplaces were grouped in compartments of up to four workplaces which related to rooms in traditional offices. The compartments were grouped in two areas which are part of the entire smart environment. This structure provided clues on where to place Fog Nodes and aggregate data of sensors and actuators.

Development: The project was implemented during the iPraktikum at the Technical University of Munich in the winter semester 2018/2019 with the Intelligent Workplace researchers as customers[6]. The iPraktikum is a multi-project practical course with up to 80 student developers who work in up to twelve project teams each semester. In each project, the developers create an application for real customers from industry and academia to solve a real problem with a mobile context. Each project team is supported by a coach, who participated in the course before and helps the students with organizational tasks as well as technical challenges, and a project lead, who is a doctoral candidate. "While the mobile context is realized using Apple's iOS platform, which results in iPhone and iPad applications, most projects are not standalone solutions, but include application servers, sensors, actuators, or wearable devices" [75]. A more detailed description of the iPraktikum can be found in Bruegge et al. [22] and Xu et al. [152].

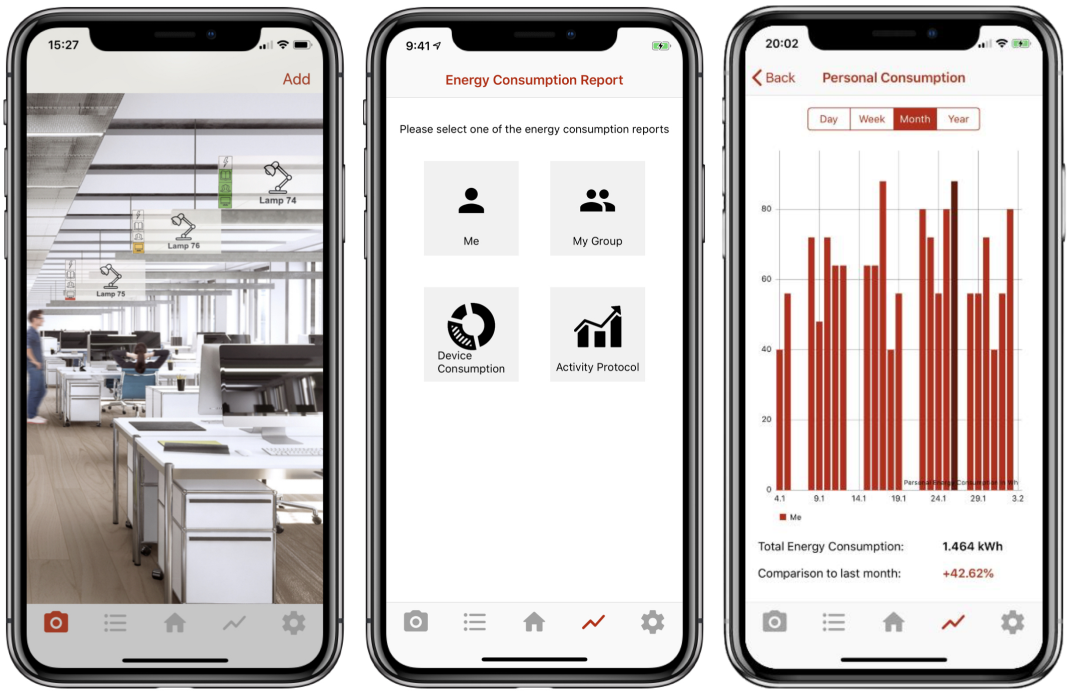
Sample Application: The resulting mobile application allowed occupants of smart environments to control their devices and get information on sensors using augmented reality. This approach addressed the goal to support users with difference in knowledge and involvement. Occupants did not have to know which devices could be controlled and did not have to know the devices' names. The augmented reality view presented all devices the occupant can interact with and abstracts away unnecessary names. Thus, even temporary occupants that rarely joined the smart environment could intuitively interact with the devices. To enable the augmented reality view, one occupant had to setup the scene by scanning their environment, placing the virtual objects, and connecting them to the according device.

[6] https://ase.in.tum.de/lehrstuhl_1/component/content/article/106-teaching/994

This data was stored in environment models and was updated whenever an occupant used the augmented reality feature. The stored data contained feature points to detect the surrounding area as well as anchors for the placed devices. To improve the accuracy and speed of the environment models and to introduce the possibility to share them, the environment models were hyper-local and usually only covered one room. Based on the hyper-locality and the need to share the environment models, the models were stored on the Fog Node corresponding to the current Fog Horizon of the occupant. When a new occupant joined the Fog Horizon of that Fog Node, they requested the current environment model from the Fog Node and could immediately interact with smart devices without the need to setup everything on their own. During usage, the environment model was constantly updated on the occupant's phone and sent to the Fog Node utilizing the low latency provided by Fog Computing.

To address the goal to raise energy awareness, the application incorporated an energy consumption report that showed occupants their energy consumption and provided the option to compare it with other occupants. Whenever an occupant turned on a device, the energy consumption of that device was added to the occupant's total consumption and stored in the cloud, enabling persistency and access to the energy consumption from outside of the device's Fog Horizon.

Figure 6.3 showcases the ARControl app. The first screen shows the augmented reality view which is an overlay on the mobile device's camera feed. Each box shows devices the occupant can interact with. By tapping on the box, the device is turned on or off, swiping allows scaling of the value, e. g., dimming of lights. To support energy saving, predefined light levels can be activated by tapping on the icons at the left of the box with the icons describing different use cases. The second and third screen show the energy consumption feature. The overview allows to change between personal consumption (*Me*), the consumption of a group the user belongs to (*My Group*), an overview of which of the user's devices consumes which amount of energy (*Device Consumption*), and the last recorded activities (*Activity Protocol*).



Total Energy Consumption: 1.464 kWh
Comparison to last month: +42.62%

(a) Augmented Reality (b) Energy Overview (c) Personal Consumption

Figure 6.3: The augmented reality control screen (a) and the energy consumption feature (b), (c) of the ARControl app. The first screen allows to interact with the environment using augmented reality, and therefore even supports users that rarely join the smart environment. The second and third screen enable to track the energy consumption of the individual user.

6.1.2 Lassie

The second case study is called *Lassie*. It contains parts from both domains, the health domain as well as the smart environment domain. The smart environment domain is particularly fitting for Fog Computing applications and dynamic approaches due to the constant change of occupants as well as devices.

Context: The health domain is getting more and more related to the smart environment domain. Smart watches and fitness trackers allow constant monitoring of vital parameters of patients, but also sensors within the patient's home provide clues on the patient's health. With more devices being used to track patient's health in addition to providing environmental comfort, these domains are interconnected. Therefore, the health domain in private homes supports the hierarchical structure of smart environments which can be used to aggregate data for each room, for each floor etc.

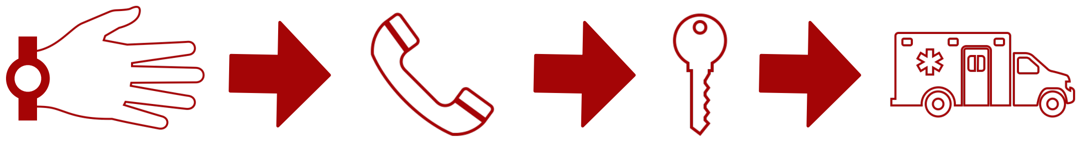


Figure 6.4: The current process of elderly people receiving help in case of an emergency. First, the patient has to signal an emergency by pressing the button on the wristband, second the emergency call is received in a call center from which an emergency worker is sent to acquire the key for the home of the elderly person from a central key management before driving to the emergency location.

Stakeholders: The stakeholders in this domain were patients that needed to get help in case of emergencies, but also emergency workers that could get notified faster and get information on the patients vital parameters before arriving at the patient's home. Thus, the emergency workers are better prepared for the case at hand.

Goal: The goal of this case study was to create a system that enhances the process of detecting emergencies for elderly people, sending help, getting access to the patient's home, and finally helping the patient. This should be achieved by using sensors for emergency detection mechanisms, Fog Computing to enable data preprocessing, and smart locks to provide access for emergency workers.

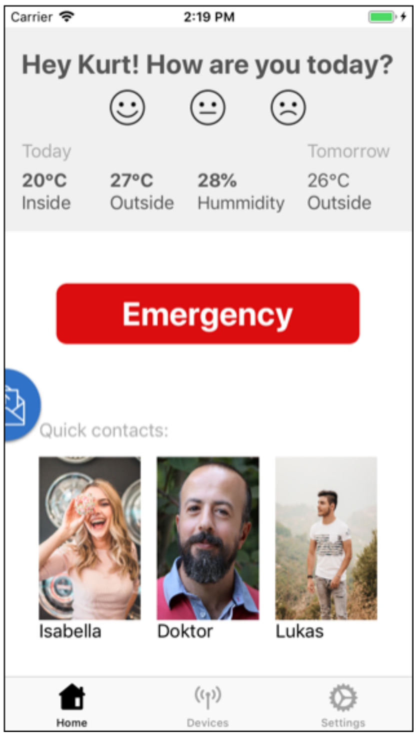
Development: During the development, we modeled the dynamic addition of new sensors as well as the elderly person leaving and rejoining the setting. Lassie was developed during another instance of the iPraktikum in the summer semester 2019[7].

The previously used process, as depicted in Figure 6.4, involved the patient wearing a wristband with a red button to call for help, an emergency call center sending the address to the emergency worker who had to acquire the key for the particular home from a central key management unit before they could access the patient's home.

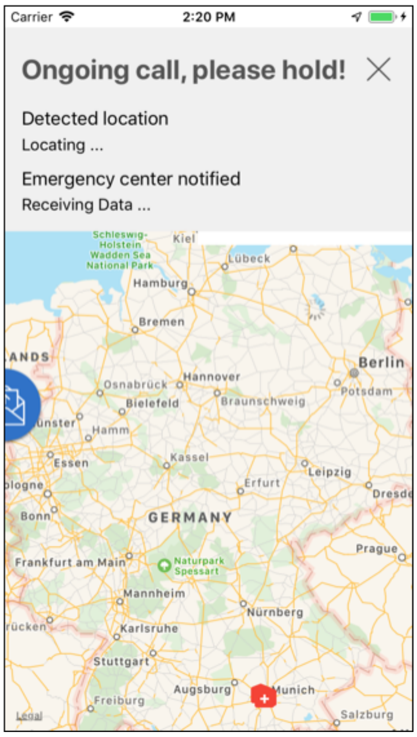
Sample Application: The developed system consisted of several parts: a mobile application tracking the elderly person's behavior using sensor data, a smart watch detecting heart rate and providing fall detection, and an emergency management system. The emergency management system consisted of a web view for the emergency call centers from the DRK (German Red Cross) to deploy emergency work-

[7]https://ase.in.tum.de/lehrstuhl_1/component/content/article/106-teaching/1037

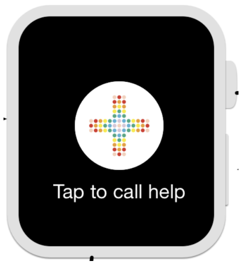
ers and an anomaly detection system that observed changes in the behavior of the elderly person. In the web view, the emergency call centers were also able to unlock the doors of elderly persons' homes in case of emergencies to provide access to emergency workers.



(a) Home Screen with emergency button



(b) Initiated Emergency Call locating the patient's position



(c) Watch App for Fall Detection and Emergency Calls

Figure 6.5: The home screen (a), the active emergency call screen (b), and the watch app (c) of the Lassie app. The home screen allows to manually trigger emergency calls, quickly contact the most important contacts, and answer questions to improve the accuracy of the anomaly detection. The watch app allows to trigger emergencies, but also enables fall detection.

Figure 6.5 showcases the resulting mobile applications. The first screen is the home screen of the application and most importantly allows the user to file emergency requests. Additionally, the user answers questions on a daily basis related to detected anomalies to improve the accuracy of the anomaly detection and can quickly access their most important contacts such as siblings, children, doctor etc. The second screen shows an emergency call in progress. While the emergency call center is contacted to send help, the user's location is identified and the shortest route from the emergency center to the user is calculated. Live updates show the user the current position of the emergency worker. To better prepare the emergency worker for the emergency, additional data such as the heart rate is send to the emergency center. The last screen shows the watch app which provides data such as heart rates, allowing the user to access the emergency call button, and detect falls.

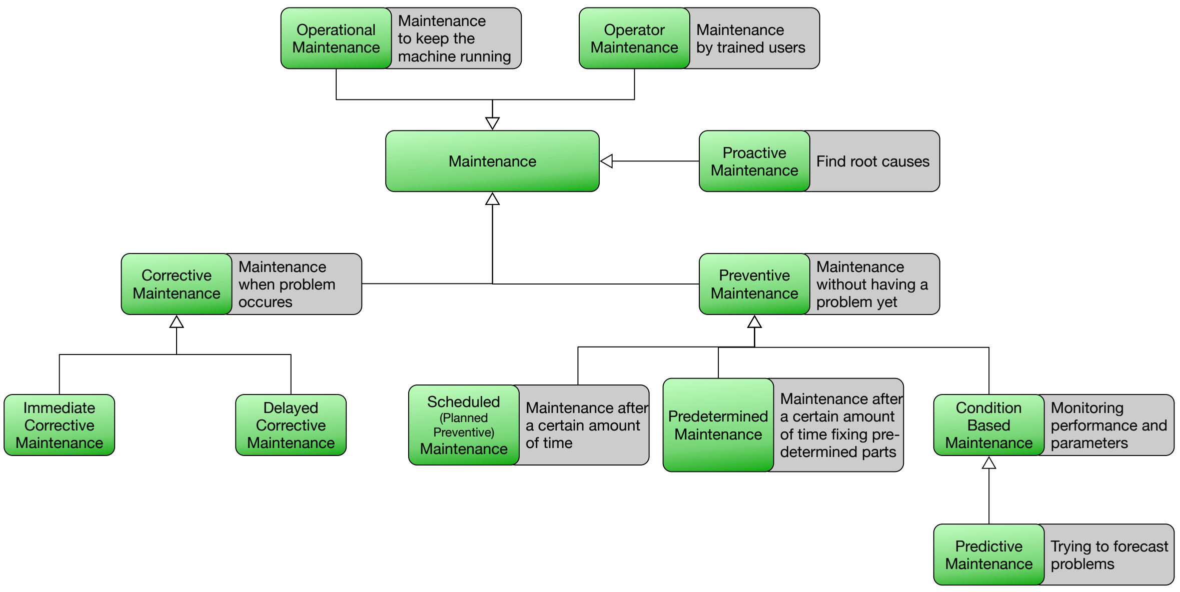

Operational Maintenance
Maintenance to keep the machine running
Operator Maintenance
Maintenance by trained users
Maintenance
Proactive Maintenance
Find root causes
Corrective Maintenance
Maintenance when problem occures
Preventive Maintenance
Maintenance without having a problem yet
Immediate Corrective Maintenance
Delayed Corrective Maintenance
Scheduled (Planned Preventive) Maintenance
Maintenance after a certain amount of time
Predetermined Maintenance
Maintenance after a certain amount of time fixing pre-determined parts
Condition Based Maintenance
Monitoring performance and parameters
Predictive Maintenance
Trying to forecast problems

Figure 6.6: An overview of the different *Maintenance* types with a short description for each to support the classification of PdMFrame.

6.1.3 PdMFrame

The third case study is called *PdMFrame*[8] which stands for ”A **Pred**ictive **M**aintenance **Frame**work” and was placed in the metrology domain.

Context: The difference between smart environments in general and the industrial domain are the used IoT devices. While in smart environments any smart device can be integrated, the industrial domain relies on reliability and connectivity [125]. Accordingly, IoT devices can be distinguished between consumer IoT devices (CIoT) and industrial IoT devices (IIoT) [125]. IIoT devices bridge the gap between the physical, industrial world and the digital environment to create a cyber-physical setting [47]. According to Scheuermann, these devices can be sensors, actuators and interactive devices [122]. Their main purpose in the industrial domain is to collect and process data for pattern recognition, prediction and optimization, e. g. maintenance [38]. This timely and efficient handling of equipment failures was already discussed by Liebetrau and Grollmisch in [88] and is essential to the prospect of the business. Equipment failure can harm the business’s reputation, is expensive, and time consuming [11, 28].

Stakeholders: The stakeholders were workers interacting with the machine, the owner of the machine, and the maintenance workers who try to keep the machine in a working state. The owner’s and worker’s interest overlap as both want to sim-

[8] Parts of PdMFrame were developed as part of a research project in collaboration with Maximilian Opbacher [104].

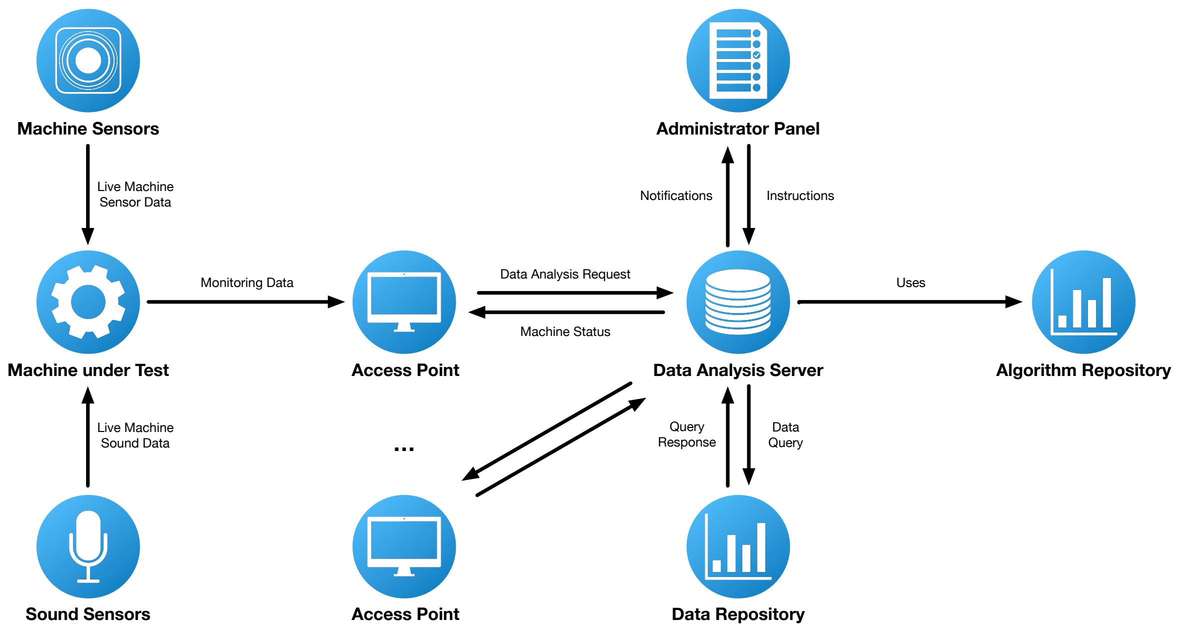


Figure 6.7: An overview of the PdMFrame system. It shows the `Machine under Test` on the left which is connected via an `Access Point` to the `Data Analysis Server`.

plify and optimize the interaction with the industrial machine which is achieved by automation, keeping the machine in a working state, and improving the human-machine-interface. The maintenance worker's interests are on predicting machine failures and, in case of an error, receive clues on the issue to be able to faster repair the machine.

Goal: The goal of PdMFrame was to use Fog Computing to gather sensor data, to aggregate the data, and to predict machine failures. The failures should be used to further improve the predictions and to predict failures on other machines of the same type. Figure 6.6 provides an overview of the different maintenance types. Accordingly, based on its goal, PdMFrame used predictive maintenance.

Development: The project was promoted by the Software Campus[9] initiative, a program by the BMBF (Bundesministerium für Bildung und Forschung). The project was developed with support of student researches. The developed solution allowed to upload input data from different sensors, such as audio, pictures, or temperatures. In the setup as shown in Figure 6.7, the machines under test were the Edge Devices with all connected sensors.

The Fog Nodes were computational units in close proximity to the machine. For each batch of sample data, they preprocessed and aggregated the collected data to reduce the network load.

[9]https://softwarecampus.de

On the cloud, the collected data was persisted and used to train a predictive maintenance model. The cloud contained an algorithm repository with different machine learning techniques. During training, the algorithms were used to create different models for prediction and they were compared with each other to find the best fitting model for the data type of the sensor. Finally, the models could be used to evaluate future values from the data source and predict machine failures.

Sample Application: One instance of this system called *AudioForesight*[10] and is presented in the paper "AudioForesight: A Process Model for Audio Predictive Maintenance in Industrial Environments" [65].

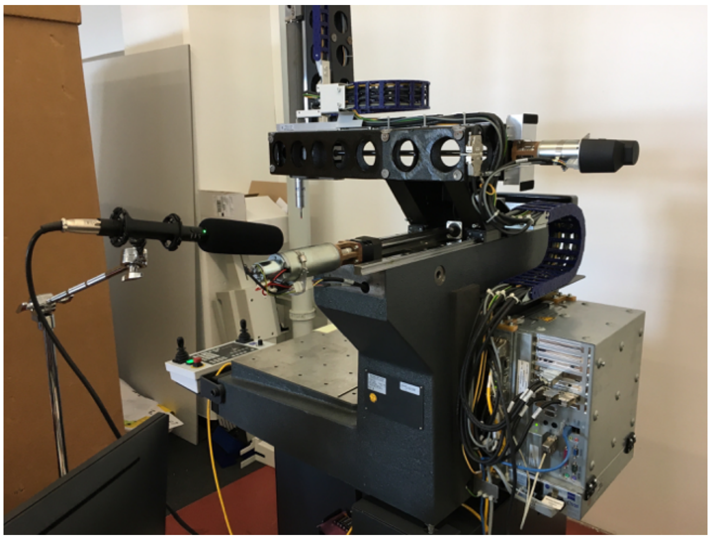


Figure 6.8: The experimental setup for the *PdMFrame* case study. The image depicts a directed microphone (on the left) pointing in the direction of one drive of the ZEISS DuraMax measuring machine.

We created a process model for audio predictive maintenance and tested the system for a ZEISS DuraMax measuring machine with a microphone to record audio samples as shown in Figure 6.8. The process model describes how those audio samples can be used to detect machine failures and integrate user feedback into the learning process of the failure detecting models using a five step approach. Figure 6.9 shows the *Audio Anomaly Detection & Classification Process Model*.

The first step, called *Data Preprocessor* encapsulates the preprocessing that can already be accomplished in a decentralized fashion, ideal for Fog Computing. Second, the *Model Trainer* uses the preprocessed data and trains two models: An anomaly detection model to recognize anomalies within the machines routine, and a classification model to provide insights into the failures in case of an anomaly.

[10]AudioForesight was developed as part of a research project in the context of PdMFrame in collaboration with Klaidi Gorishti [56].

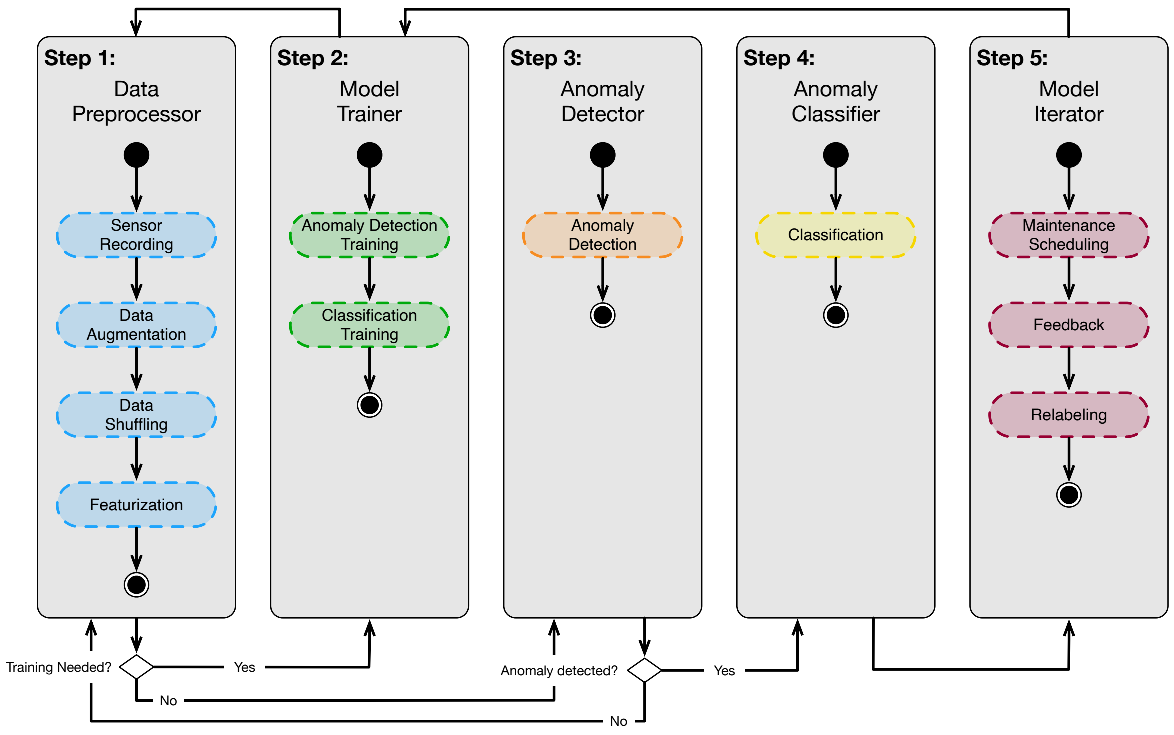


Figure 6.9: The *Audio Anomaly Detection & Classification Process Model* for AudioForesight adapted from Henze et al. (UML state chart diagram, © 2019 IEEE) [65]. The process is divided into five steps highlighted by different colored boxes.

Third, as soon as the models are trained, the *Anomaly Detector* uses newly incoming sensor data to detect anomalies. In case an anomaly is detected, step four, the *Anomaly Classifier* tries to identify the failure at hand to provide the maintenance worker with hints on what went wrong. These failure classes are specific to the industrial machine. Finally, in step five, the models are retrained based on the feedback the maintenance workers provided on the questions if the detected anomaly was a true positive and if the failure class was helpful.

6.1.4 DisCoFog

The fourth case study can be separated into two system instantiations: *DisCoFog 1* and *DisCoFog 2*. Both instances are described as part of the case study in "Fog Horizons – A Theoretical Concept to Enable Dynamic Fog Architectures" [66]. DisCoFog is short for "**Dis**covery and **Co**nnectivity Protocol for **Fog** Environments".

Although the instances are linked to the two domains smart cities and smart environments, the concept of DisCoFog itself is domain-independent. For this general case, the stakeholders are the component developers as well as the software architects. The component developers are interested in establishing connections to other components which offer services that the developers want to use for their component. Thus, they need to be able to find those components that are in close proximity and offer specific services. The software architects on the other hand are interested in the entire architecture, which components are involved and their connectors as specified by the software architecture definition in Section 3.1.1.

Therefore, the goal of DisCoFog is to create a simple discovery and connectivity protocol based on multi-casts to find other nearby Fog Components for services specified by the service consumer. The entirety of the discovery process and the resulting connections, represent the final Fog Architecture setup.

DisCoFog 1

DisCoFog 1[11], the first instance of the DisCoFog protocol, is based in the infrastructure domain; more precise, a smart city with multiple cities and drones traveling in between them. Although presented using drones, the smart city domain representes an ideal target domain for the application of Fog Computing and the scalable aspects of xFogPlus. First, the diversity of potential Fog Components ranges from, e. g., cars, bicycles, pedestrians, and traffic lights up to servers as coordinators for traffic flow. Second, the hierarchical structure of cities describes layers that promote locality. For example, cars report and get information to/from traffic lights, those are grouped into streets, streets into city districts, districts into cities, and so on–grouping parts that belong together within a local proximity while moving up the hierarchy.

Stakeholders: The smart city domain includes different stakeholders. The traffic participants want to get from A to B in the least time possible, avoiding traffic, and accidents. On the other side, we have emergency workers that help in case of accidents; construction side workers; people regulating traffic, changing traffic lights signals; politicians approving beltways; and many more.

[11]DisCoFog 1 was developed as part of a research project in collaboration with Paul Schmiedmayer [123].

Their overall interest is to enable fast transportation while keeping the traffic participants save. Therefor, they need data about the traffic density, the location of accidents, and construction sites, etc.

Goal: The goal of DisCoFog 1 was to use the simple discovery and connectivity protocol to propagate the needed information to the corresponding stakeholders.

Development: The first emphasis in DisCoFog 1 was the development of the DisCoFog protocol itself which was user data protocol (UDP)-based [111] and used internet protocol (IP)-based multicasting to evaluate the Fog Horizon of a Fog Component. Therefore, each Fog Component listened on a predefined port on a multicast address. The dynamic model of the message exchange is displayed in Figure 6.10 and shows *Edge Device E*, an example service consumer, requesting *Service X*. *Fog Node A* and *Fog Node B* which offer this service respond with their service information, while *Fog Node C* which only offers *Service Y* does not. After selecting a service provider, in this case *Fog Node A*, based on the fastest response time, the connection is established using the service information provided and data can be exchanged. Which service provider answers the fastest is based on several aspects: The local proximity of the service provider to the service consumer, the workload of the service provider, as well as the network connection and communication channel between service consumer and service provider.

Sample Application: The DisCoFog 1 setup described a multi-city approach that used drones to transport goods between them. The drones, using their sensors and actuators, flew autonomously from one city to the next city and tried to avoid collisions. The drones were Phantom 4[12] by DJI[13] which can autonomously fly using a smartphone application that was created during JASS 2018[14] with adjustments to GPS. JASS, short for **J**oint **A**dvanced **S**tudent **S**chool, is a collaboration between Saint Petersburg Electrotechnical University and Technical University of Munich in which students from both universities meet and collaborate for two weeks on a project with changing subjects. The goal of JASS 2018 was the creation of a multi-modal transportation use case using drones and self-driving cars.

The drones used the city servers as a Fog Component that offered location data of all connected drones to each other. Additionally, this location data was gathered from all cities on a central cloud to calculate 3D heat maps of the drone movement to find the most common routes and endangered areas.

[12]https://www.dji.com/de/phantom-4-pro-v2
[13]https://www.dji.com
[14]https://ase.in.tum.de/lehrstuhl_1/projects/all-projects/973-jass-2018

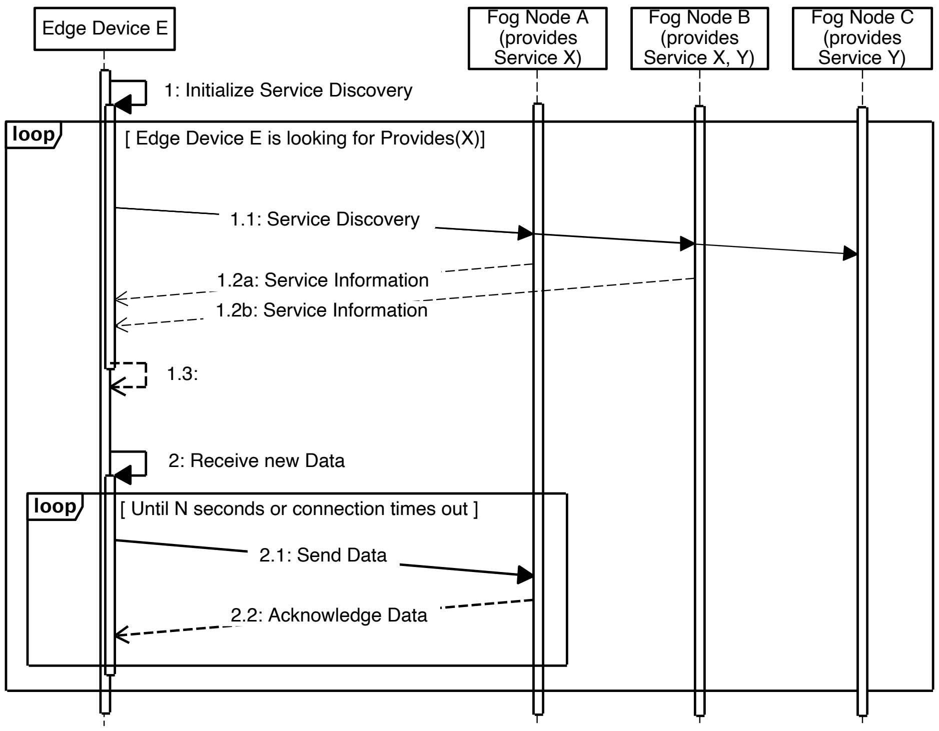

Edge Device E
Fog Node A (provides Service X)
Fog Node B (provides Service X, Y)
Fog Node C (provides Service Y)
1: Initialize Service Discovery
loop
[Edge Device E is looking for Provides(X)]
1.1: Service Discovery
1.2a: Service Information
1.2b: Service Information
1.3:
2: Receive new Data
loop
[Until N seconds or connection times out]
2.1: Send Data
2.2: Acknowledge Data

Figure 6.10: A dynamic model describing the message exchange of the DisCoFog protocol for a potential service consumer (*Edge Device E*) and three service providers (*Fog Node A, B, C*). The requested service is *X* which is offered by *Fog Node A, B* (UML Sequence Diagram, © 2019 IEEE) [66].

Each drone continuously evaluated its Fog Visibility as well as Fog Horizon using the DisCoFog protocol to avoid collisions. When leaving a city's outreach, and thus loose the connection, they tried to find a new communication partner offering the same service.

DisCoFog 2

DisCoFog 2[15] is the second instantiation of the DisCoFog protocol and placed in the smart environment domain. As previously discussed in Section 6.1.1, this domain is particularly fitting for Fog Architectures, but also for scalability due to its inherent hierarchical structure. DisCoFog 2 is placed in the same environment as ARControl. For both, the Intelligent Workplace [63] was the target domain. While ARControl focused on the dynamics of occupants and establishing intuitive controls for all occupants no matter the amount of time they spent in the environment, DisCoFog 2 focused on establishing a scalable Fog Architecture setup, while having the same occupant dynamics.

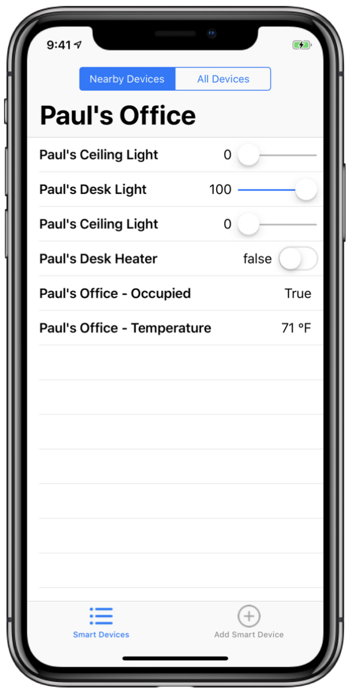
[15]DisCoFog 2 was developed as part of a research project in collaboration with Paul Schmiedmayer [124].

Stakeholders: Accordingly, the occupants of the smart environment were stakeholders interested in interacting with the available devices. Other stakeholder groups were developers, maintainers, and system administrators of the smart environment. Besides providing the controls to the occupants, they were interested in the consistency of the overarching architecture of the smart environment, the grouping of logically connected devices, and the discovery of those devices.

Goal: DisCoFog 2 tried to address these interests with the goal of creating a Fog Architecture based on discovered services, logically connected devices, e. g., all devices in one office are connected to one Fog Component controlling everything within the office, and allowed occupants to discover the devices based on their current Fog Horizon.

Development: First, we established the architecture: Every device which did not rely on a centralized control system was connected to decentralized Fog Components according to the office layout. In contrast, many devices were not directly addressable and could only be used with a proprietary communication standard and a central communication server. The offices could be aggregated to bigger units, e. g., a Fog Component which controlled several offices or a Fog Component that controlled an entire floor. Each building contained one central cloud instance that could be reached from outside of the smart environment which created reports on power consumptions and the current states of devices. This approach allowed to investigate the scalable concepts of xFogPlus.

Sample Application: The second part of DisCoFog 2 was a mobile application which utilizes this decentralized approach. The application is shown in Figure 6.11. It used the DisCoFog protocol to connect to the Fog Component closest to the occupant. Afterwards, it displayed all available devices the user could interact with in the current surrounding which relates to the occupant's Fog Horizon. Additionally, if no Fog Component was nearby which the occupant's smartphone could connect to, the app accessed the central cloud instance to interact with all devices. Figure 6.11 shows three screens of the application while the occupant walks from *Paul's Office*, through the *Kitchen*, to the *Conference Room*. Each screen shows the corresponding devices the occupant can interact with in the current Fog Horizon. The first screen shows a list of all devices the occupant can interact with in *Paul's Office*, the second screen the devices in the *Kitchen*, and the third screen the devices in the *Conference Room*.


9:41

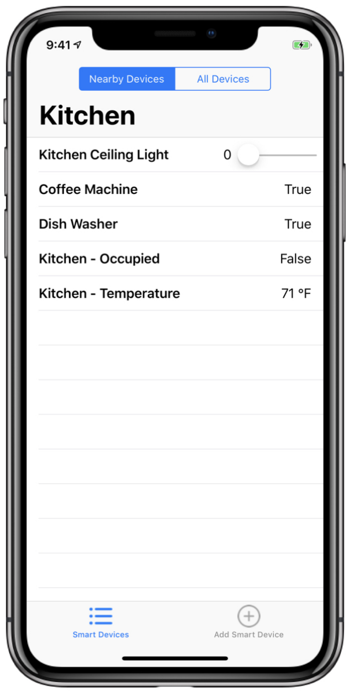
Nearby Devices | All Devices

Paul's Office

Paul's Ceiling Light	0
Paul's Desk Light	100
Paul's Ceiling Light	0
Paul's Desk Heater	false
Paul's Office - Occupied	True
Paul's Office - Temperature	71 °F

Smart Devices | Add Smart Device

(a) List of the devices accessible in *Paul's Office*.


9:41

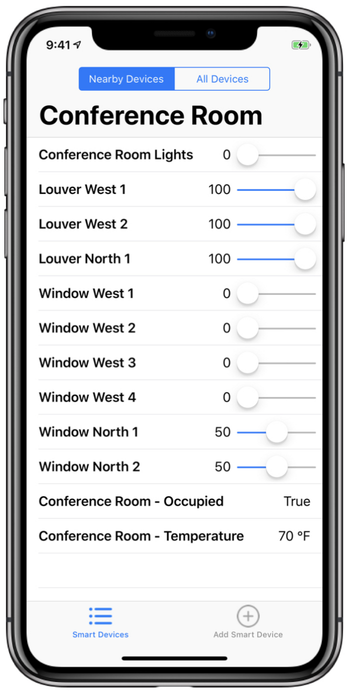
Nearby Devices | All Devices

Kitchen

Kitchen Ceiling Light	0
Coffee Machine	True
Dish Washer	True
Kitchen - Occupied	False
Kitchen - Temperature	71 °F

Smart Devices | Add Smart Device

(b) List of the devices accessible in the *Kitchen*.


9:41

Nearby Devices | All Devices

Conference Room

Conference Room Lights	0
Louver West 1	100
Louver West 2	100
Louver North 1	100
Window West 1	0
Window West 2	0
Window West 3	0
Window West 4	0
Window North 1	50
Window North 2	50
Conference Room - Occupied	True
Conference Room - Temperature	70 °F

Smart Devices | Add Smart Device

(c) List of the devices in the *Conference Room*.

Figure 6.11: The DisCoFog 2 mobile control application. The application uses a Fog Architecture and the DisCoFog protocol to provide access to all nearby devices, as those are used most often by occupants. While walking through the smart environment the *Nearby Devices* tab adjusts to the current Fog Horizon and lists all nearby devices the occupant can interact with. The *All Devices* tab shows all devices instead.

During the development of DisCoFog 2, we refactored the existing lightweight DisCoFog protocol to use established standards. We selected mDNS- and DNS-based service discoveries [30]. Additionally, we used DNS messages [98], DNS SRV records [61] and DNS-SD TXT records [98] to encode the service requests.

6.1.5 eHealth

The fifth case study is called *eHealth*[16] and is one of the case studies for the *Scalable Fog Architectures* validation.

Context: The case study was placed in the health domain, in particular in a hospital setting. As already described in Section 6.1.2, the health domain is well suited for Fog Computing due to the amount of different devices used. Additionally, the hospital setting with different treatment rooms are also fitted for the scalable aspects of xFogPlus. The treatment rooms were filled with medical devices as well as PCs, which in turn belonged to different departments that could be on different floors. In this setting, the locality aspect of Fog Computing was not only helpful to provide computational power closer to the edge to process the massive amounts of data newer medical device gather, but also prevented the need to send private patient data throughout the entire hospital network.

Stakeholders: Three stakeholders stand out: The patients who wanted to have optimal treatment, while knowing their private information being secured, the system administrators, who were responsible for the network's security, and the doctors, who wanted to access the patient's information to apply the optimal treatment.

Goal: Thus, the goal of eHealth was to create a system that integrates Fog Computing and especially the scalable aspects of xFogPlus in a hospital setting to ensure optimal treatment for patients by providing the needed computational power for modern medical equipment closer to the devices. Additionally, the privacy aspect should be investigated.

Development: eHealth focus was set on providing a scalable architecture and ensuring privacy among different layers. The developed platform offered the possibility to define access policies that specified which user was allowed to access which data of which patient.

Sample Application: The first application within this system was an image server that persisted medical images and allowed other services to request them. These images were stored in the **D**igital **I**maging and **Co**mmunications in **M**edicine format (DICOM), an international standard for storing, transmitting, and displaying medical information [16, 97, 101]. The second service was the counterpart to the image server, a DICOM viewer which used WebSockets to request the files.

[16]eHealth was developed as part of a research project in cooperation with Philipp Diller [42].

Finally, a third service simulated the mobile devices of doctors that joined the hospital network on application startup and left the network when the application was closed. To encrypt the information, we selected the attribute-based message encryption protocol *FAME* as introduced by Agrawal et al. [1].

6.1.6 Fog.BOI

The six case study is a scalable **B**roker **O**perations and **I**nteraction system for dynamic and scalable **Fog** Computing. Fog.BOI is not domain-specific, but rather a concept that uses the scalable aspects of xFogPlus.

Stakeholders: Therefore, the stakeholders are identical with the stakeholders or actors of xFog. First, the software architect is interested in providing higher-level services to layers that are not adjacent. This enables software architects to tunnel services to lower layers without using additional computational resources or storage capacities. Second, the component developers can provide services they use themselves to implement the functionality of the component they are developing to their own service consumers. Additionally, messages can be aggregated and filtered, as well as propagated through the network.

Goal: The goal of Fog.BOI was to use the scalable concepts of xFogPlus and provide mechanisms to propagate and broadcast messages up and down a hierarchical Fog Architecture. In this context, propagating messages referred to the forwarding of messages up the hierarchy; Broadcasting messages meant to send messages to every service consumer of the current service provider, and therefore spread information down the hierarchy.

Development: Fog.BOI[17] was first described in the context of a smart city environment to propagate traffic information throughout the hierarchical structure. Thus, Fog.BOI was an instance of the smart city domain, similar to the first DisCoFog instance, as shown in Figure 6.1.

Sample Application: It was a message broker system that used socket-based communication. Each Fog Component represented one message broker that provided an arbitrary number of channels for other Fog Components on lower layers to subscribe to. The message brokers created message handlers to define custom interactions. Additionally, Fog Components subscribed to channels of parent layers. If desired by both component developers, services from higher layer Fog Components could be consumed but also promoted by lower level Fog Components.

[17]Fog.BOI was developed as part of a research project in cooperation with Sebastian Aigner [2].

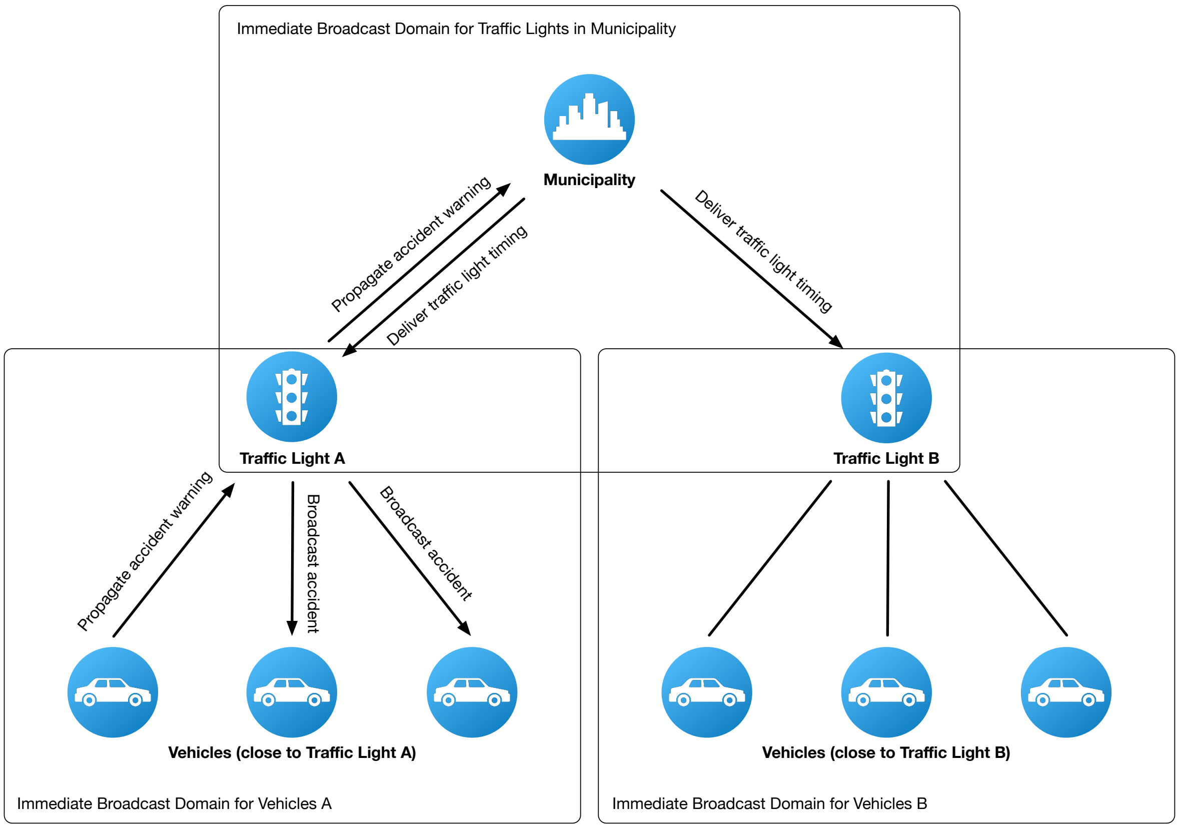

Immediate Broadcast Domain for Traffic Lights in Municipality
Municipality
Propagate accident warning
Deliver traffic light timing
Deliver traffic light timing
Traffic Light A
Traffic Light B
Propagate accident warning
Broadcast accident
Broadcast accident
Vehicles (close to Traffic Light A)
Vehicles (close to Traffic Light B)
Immediate Broadcast Domain for Vehicles A
Immediate Broadcast Domain for Vehicles B

Figure 6.12: An example setup for the infrastructure domain. In this 3-layered Fog Architecture, all traffic participants are on the Edge Layer and share information about accidents they are involved in or detect with the traffic light within its Fog Horizon. This information is broadcasted to the other traffic participants to avoid congestions. Additionally, the information is aggregated and propagated to the traffic lights parent node, the municipality, which change the traffic lights' behavior to guide the traffic around the accidents.

Thus, the number of services might increase the lower the layer of the Fog Component was, but only a small fraction of those services were calculated and evaluated in the Fog Component itself.

One use case for this system was the smart city domain. As shown in Figure 6.12, traffic lights offered an accident warning service. This service collected the location data of traffic participants in the local surrounding described by the Fog Horizon if they were involved in or reported an accident. Thus, every traffic participant could receive the locations of accidents and avoid the respective street. Higher-level Fog Components provided channels in which traffic lights could propagate this accident data in an aggregated form to adjust the traffic lights, and therefore guide the traffic around the accident. To enable this behavior, Fog Components, in form of traffic participants, joined and left the environment at runtime.

6.1.7 Quasar

Quasar[18] represents the seventh case study. In comparison to the previous case studies, Quasar focused on the service provider selection introduced by the xFogStar workflow. While Fog Computing handles the process of finding potential service providers, not every service provider that offers the requested service fits the service consumers demands. Providing a service based on the consumer's needs is of crucial importance [9, 25, 26].

Context: Quasar was placed in the domain of online gaming. This domain showed strict requirements on offered services. For example, Zhang et al. discuss latency and high bandwidth demands as key requirements for online gaming [159]. Latency leads to the game "lagging" behind which can lead to bad gaming experience and even motion sickness [159]. We already discussed both of these parameters as part of the QoS vector in Chapter 5. According to Yi et al. also parameters such as connectivity, reliability, and capacity are of importance [156]. As these parameters can directly be addressed using the concept of Fog Computing, the domain fits the case study's intention.

Stakeholders: Several stakeholders are involved in the gaming domain. First, the players themselves want to have an optimal gaming experience without being interrupted by undesirable connection interruptions or latency issues. Second, the game developers want to provide the best gaming experience to the players to ensure revenue and ideally a growing player base. Third, network providers prefer local network traffic inside their own infrastructure over routing huge data amounts to other internet provider's servers.

Goal: Accordingly, the goal of this case study was to instantiate the workflow for service provider selection as described by the xFogStar workflow in the gaming domain to find the best fitting service provider within a given Fog Architecture. Introducing the locality of Fog Computing addressed the internet provider's interests as well as faster latencies for the players, while the selection workflow addressed the player's and game developer's intentions.

Sample Application: The developed mobile application to validate the service provider selection was the game as presented in Figure 6.13. The screen of the smartphone mirrored a gameboy[19] and allowed the player to interact with the scene, shown at the top of the screen, using the displayed buttons. Before the game started, the player had to choose the server to connect to.

[18]Quasar was developed as part of a research project in collaboration with Sandra Grujovic [59].
[19]https://de.wikipedia.org/wiki/Game_Boy

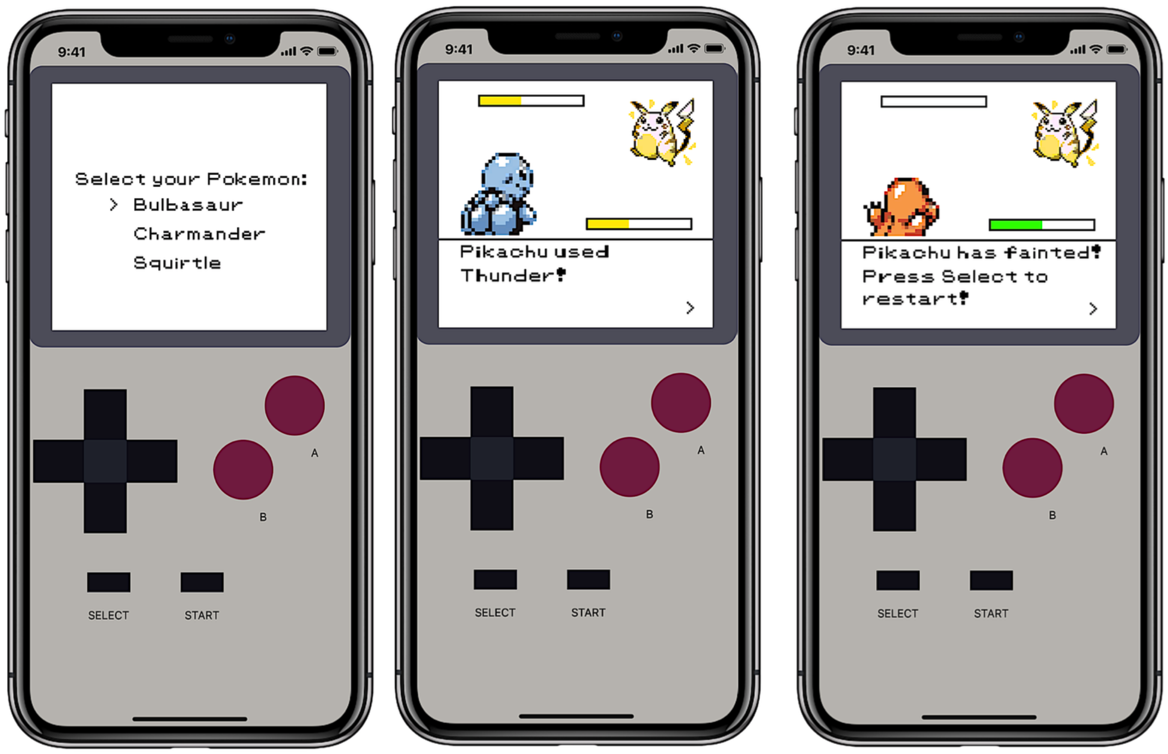


Figure 6.13: A game for testing the service provider selection workflow. On the first screen, the player can choose which character they want to play. The second screen shows the engagement with another player or AI and the third screen shows the end of the game.

Each scene was created remotely and streamed as a video feed to the smartphone. Each action needed to be send to the remote server to trigger the next state of the game, thus requiring a good latency and bandwidth to accomplish an enjoyable gaming experience. The game also provided multiplayer capabilities. Whenever two players selected the same game service provider, they were matched against each other. The server then calculated two scenes for the same game. On one scene, the first player was visualized as the main player and the second player as the enemy, on the second scene, it was the other way around. These scenes were send to the according player.

6.1.8 FoQsIs

The second case study for the service provider selection is called *FoQsIs*[20]. FoQsIs is short for **<u>Fo</u>**g Component based <u>Q</u>o<u>s</u> discovery for continuous <u>I</u>ntegration<u>s</u>.

Context: In this case study, xFogCore and xFogPlus handled the service discovery and dynamic aspects. We focused on the service provider selection by instantiating the workflow of xFogStar in the domain of continuous integration.

> "Continuous Integration is a software development practice where members of a team integrate their work frequently [...]. Each integration is verified by an automated build [...] to detect integration errors as quickly as possible."
>
> — MARTIN FOWLER [49]

The quote by Fowler describes the main idea of continuous integration. Thus, continuous integration requires at least one remote server to perform builds and tests. Further techniques, e. g., continuous delivery, are directly linked to continuous integration and often used simultaneously, as shown by Krusche et al. [83, 84] or Klepper et al. [79].

Stakeholders: The stakeholders of the continuous integration domain are developers working in the same team and the project leaders. The developers want to reduce the effort of integrating their changes by pushing small changes to the server. The project leaders' interest is the reduced time it takes to solve small integration problems and the reduced amount of errors that occur.

Goal: The goal of this case study was to combine Fog Computing and continuous integration. We used Fog Computing to find available integration servers in close proximity and evaluated the performed builds of all servers on a cloud instance. Thus, we allowed decentralized builds for developers supporting the idea of continuous integration, and provided an overview of the different integration servers on the cloud.

Sample Application: The developed application, called *FoQsIs Client*, could be deployed on a smartphone or a macOS[21] device. It used mDNS- and DNS-based service discoveries to find the best fitting integration server based on the user's needs and followed the xFogStar workflow.

[20]FoQsIs was developed as part of a research project in collaboration with Philipp Eichstetter [45].
[21]https://www.apple.com/de/macos/what-is/

6.2 Results

In this section, we present the results of the case studies. Also having real applications for the concepts of xFogCore and xFogPlus which are of theoretical, modeling nature, we mainly address how the concepts translate into the models of the case studies and which benefits they provide. To show a more detailed overview on the different Fog Architectures compared to those used to introduce the concepts in Chapter 3-Chapter 4, we present a simplified hardware / software mapping (HW/SW mapping) for each case study. In this simplified HW/SW mapping, for readability, we include the components themselves, but exclude the environments the components are executed on and the according operating systems. This allows us to present even larger architectures with manageable complexity and without unnecessary overhead which is not of interest for the concepts. Fog Components on the Fog Layer are for convenience executed on Raspberry Pis[22] if not stated differently.

In the results for xFogStar, we use a HW/SW mapping to provide an overview of the involved components. We present the instantiated workflow and the resulting selection possibilities in the respective domain.

6.2.1 Dynamic Fog Components

The first results are from the *Dynamic Fog Components* validation. In this validation, we addressed the dynamic behavior of Fog Components. Accordingly, we present the changes to the models, respectively the architectures, when *new* Fog Components are added to the Fog Architectures. We show the properties of the added Fog Components in accordance to Section 4.1.1 and their placement within the architecture as described in Section 4.1.2. Therefore, we evaluate the Fog Set and Communication Set of the Fog Architecture for each case study, the service set and the resulting layer definitions, and the placement of new Fog Components within those. Finally, we elaborate on the updated Fog Architectures.

[22]https://www.raspberrypi.org

ARControl

The case study *ARControl*, as introduced in Section 6.1.1, used a 3-layered Fog Architecture approach as shown in Figure 6.14. As mentioned in Section 4.1.2, in Fog Architectures this is equivalent to tiers.

First, on the *Cloud Layer*, a central `OpenHABServer` enabled the usage of the smart environment. OpenHAB[23] stands for **Open** **H**ome **A**utomation **B**us and is an open source project for smart environment controls. It allows to integrate more than 2000 devices from different providers into a single solution, including other home automation systems. Thus, it provides a single uniform user interface for smart environment occupants to interact with and a common way of defining automation and creating rules. OpenHAB being an open source project enables customizations and the integration of decentralized control concepts, which is why we selected it as the central smart environment instance.

On the second layer, the *Fog Layer*, we introduced three Fog Nodes that control a portion of the smart environment each. The ARControl setup involved three offices which gave the Fog Nodes their names: `Paul's Office`, the `Kitchen`, and the `Conference Room`. For simplicity, each Fog Component on the Fog Layer was deployed on a Raspberry Pi[24]. Raspberry Pis are particularly suited for decentralized architectures because they provide a large amount of computational power for an affordable price and offer a multitude of communication channels. As most of the infrastructure in the Intelligent Workplace (Introduced in Section 6.1.1) was set up before WIFI was commonly used in IoT devices, we also connected the Raspberry Pis via ethernet.

Finally, the *Edge Layer* contained all available IoT devices that the occupant could interact with. Those sensors and actuators were assigned to one of the Fog Nodes on the *Fog Layer* representing the offices. `Paul's Office` included a ceiling light, a desk light, an occupancy sensor, a desk heater, and a temperature sensor. The `Kitchen` provided access to its ceiling light, an occupancy sensor, and a temperature sensor. The `Conference Room` included controls for the ceiling light, a contact sensor for each window, an occupancy sensor, and a temperature sensor.

The presented HW/SW mapping provides an insight into the smart environment concept, but for readability does not include every IoT device the occupant can interact with. For example, the `Ceiling Lights` in all offices of the Intelligent Workplace consisted of four lights which could be controlled individually or as one unit. In the `Conference Room` even more individual lights made up `Ceiling 3`. Additionally, `Conference Room` contained more windows which could not only be asked for the current status but opened and closed. In front of each window was a reflector that directed sunlight into the office or blocks it if it is too bright.

[23]https://www.openhab.org
[24]https://www.raspberrypi.org

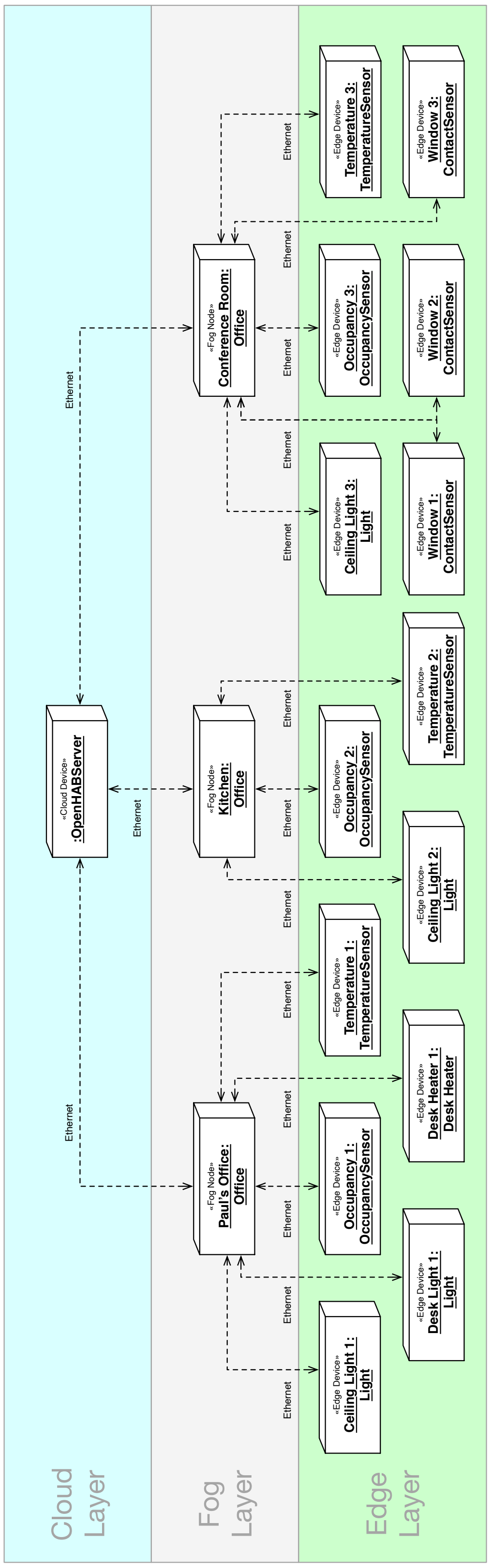

Cloud Layer
Fog Layer
Edge Layer
«Cloud Device» :OpenHABServer
«Fog Node» Paul's Office: Office
«Fog Node» Kitchen: Office
«Fog Node» Conference Room: Office
«Edge Device» Ceiling Light 1: Light
«Edge Device» Desk Light 1: Light
«Edge Device» Occupancy 1: OccupancySensor
«Edge Device» Desk Heater 1: Desk Heater
«Edge Device» Temperature 1: TemperatureSensor
«Edge Device» Ceiling Light 2: Light
«Edge Device» Occupancy 2: OccupancySensor
«Edge Device» Temperature 2: TemperatureSensor
«Edge Device» Ceiling Light 3: Light
«Edge Device» Window 1: ContactSensor
«Edge Device» Occupancy 3: OccupancySensor
«Edge Device» Window 2: ContactSensor
«Edge Device» Temperature 3: TemperatureSensor
«Edge Device» Window 3: ContactSensor
Ethernet

Figure 6.14: The HW/SW mapping of the ARControl setup (UML Deployment Diagram). For readability, we only show components and the used communication channels, but leave out operating systems etc. ARControl is a 3-layered Fog Architecture containing a *Cloud Layer*, a *Fog Layer*, and *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers.

Equation 34: ARControl: Fog Set

$$\begin{aligned}\mathit{FogSet} = \{&\mathit{OpenHABServer}, \mathit{Paul'sOffice}, \mathit{Kitchen}, \mathit{ConferenceRoom},\\ &\mathit{CeilingLight1}, \mathit{DeskLight1}, \mathit{Occupancy1}, \mathit{DeskHeater1},\\ &\mathit{Temperature1}, \mathit{CeilingLight2}, \mathit{Occupancy2}, \mathit{Temperature2},\\ &\mathit{Ceiling3}, \mathit{Window1}, \mathit{Occupancy3}, \mathit{Window2}, \mathit{Temperature3},\\ &\mathit{Window3}\}\end{aligned}$$

All of the presented IoT devices represented Fog Components based on the definition in Chapter 3 and are included in the Fog Set, as shown in Equation 34. Equation 35 shows the corresponding Communication Set of this Fog Architecture. For readability, we do not include every single Communication Component but use two simplifications to present the Communication Set: All connections between the Fog Components are bidirectional which is why, instead of listing them twice with switched Fog Components, we list them with the left-right-arrow that is used to indicate reflexivity. Second, as all Edge Devices use ethernet as their communication channel, we only list one Communication Component for each office to a pseudo Edge Device. The entire, unabbreviated Communication Set is shown in the appendix in Equation 87.

Equation 35: ARControl: Communication Set

$$\begin{aligned}\mathit{CommunicationSet} = \{&(\mathit{OpenHABServer}, \mathit{Etherent}, \mathit{Paul'sOffice})^{\leftrightarrow},\\ &(\mathit{OpenHABServer}, \mathit{Ethernet}, \mathit{Kitchen})^{\leftrightarrow},\\ &(\mathit{OpenHABServer}, \mathit{Ethernet}, \mathit{ConferenceRoom})^{\leftrightarrow},\\ &(\mathit{Paul'sOffice}, \mathit{Ethernet}, \mathit{EdgeDevice})^{\leftrightarrow},\\ &(\mathit{Kitchen}, \mathit{Ethernet}, \mathit{EdgeDevice})^{\leftrightarrow},\\ &(\mathit{ConferenceRoom}, \mathit{Ethernet}, \mathit{EdgeDevice})^{\leftrightarrow}\}\end{aligned}$$

The Service Set for our case study, as shown in Equation 36, contains six services. Of those services, the `OpenHABServer` offered three: `getGlobalDeviceList`, `getGlobalDeviceValue`, and `getGlobalDeviceValue`. `getGlobalDeviceList` returned a list of all available devices within the entire smart environment. The services `getGlobalDeviceValue` and `setGlobalDeviceValue` provided the occupant the possibility to interact with the smart environment by reading the current value of any device or setting the state of an actuator, e. g., a ceiling light. Fog Components on the Fog Layer offered three services as well: `getDeviceList`, `getDeviceValue`, and `setDeviceValue`. `getDeviceList` was the local equivalent of `getGlobalDeviceList` which returned a list of all available devices within the current office enabling locality.

The services `getDeviceValue` and `setDeviceValue` were the equivalents and were used by their global counterparts. This mapping is shown in the *Service Set Mapping*. It consists of several triples describing how the service sets are mapped to the different layers of the Fog Architecture:

$$(\{\mathrm{ConsumedServices}\}, \mathrm{Layer}, \{\mathrm{ProvidedServices}\})$$

For readability, we introduce colors that highlight the different parts of this triple: *red* for consumed services, *green* for the current layer, and *blue* for the provided services.

Equation 36: ARControl: Service Set and Service Set Mapping

$$\begin{aligned}\mathrm{ServiceSet} = \{&\mathrm{getGlobalDeviceList}, \mathrm{getGlobalDeviceValue},\\ &\mathrm{setGlobalDeviceValue}, \mathrm{getDeviceList},\\ &\mathrm{getDeviceValue}, \mathrm{setDeviceValue}\}\\ \mathrm{ServiceSetMapping} = \{&(\{\}, \mathrm{CloudLayer}, \{\mathrm{getGlobalDeviceList},\\ &\quad \mathrm{getGlobalDeviceValue}, \mathrm{setGlobalDeviceValue}\}),\\ &(\{\mathrm{getGlobalDeviceList}, \mathrm{getGlobalDeviceValue},\\ &\quad \mathrm{setGlobalDeviceValue}\}, \mathrm{FogLayer},\\ &\{\mathrm{getDeviceList}, \mathrm{getDeviceValue}, \mathrm{setDeviceValue}\}),\\ &(\{\mathrm{getDeviceList}, \mathrm{getDeviceValue}, \mathrm{setDeviceValue}\},\\ &\mathrm{EdgeLayer}, \{\}\}\end{aligned}$$

Accordingly, the layers are defined as shown in Equation 37 with the Cloud Layer being defined by the three services: `getGlobalDeviceList`, `getGlobalDeviceValue`, and `setGlobalDeviceValue`. The Fog Layer is defined by consuming the services offered by the Cloud Layer and providing the services: `getDeviceList`, `getDeviceValue`, and `setDeviceValue`. Lastly, the Edge Layer consumes the services provided by the Fog Layer.

Equation 37: ARControl: Layer Definitions

$$\begin{aligned}\mathrm{CloudLayer} &= \{\mathrm{OpenHABServer}\}\\ \mathrm{FogLayer} &= \{\mathrm{Paul'sOffice}, \mathrm{Kitchen}, \mathrm{ConferenceRoom}\}\\ \mathrm{EdgeLayer} &= \{\mathrm{CeilingLight1}, \mathrm{DeskLight1}, \mathrm{Occupancy1}, \mathrm{DeskHeater1},\\ &\quad \mathrm{Temperature1}, \mathrm{CeilingLight2}, \mathrm{Occupancy2}, \mathrm{Temperature3},\\ &\quad \mathrm{CeilingLight3}, \mathrm{Window1}, \mathrm{Occupancy3}, \mathrm{Window2},\\ &\quad \mathrm{Temperature3}, \mathrm{Window3}\}\end{aligned}$$

Table 6.1: Hard and soft requirements of Fog Components according to Section 4.1 for the dynamic Fog Component: `Occupant:Smartphone`. Hard requirements are highlighted in *red*, soft requirements in *orange*.

		`Occupant:Smartphone`
1.	Interconnectivity	✓
2.	Information Sharing	✓
3.	Uniquely Addressable	✓
4.	Computational Capabilities	✓
5.	Wireless communication	✓
6.	Locality	✓
7.	General-purpose Computational Capabilities	✓
8.	Offers Capabilities as Service	X

Using the introduced components and services, we investigated the `Occupant:Smartphone` which represented the dynamic components of this case study. It is highlighted with ① in Figure 6.15.

According to Section 4.1.1, to add a component to the Fog Architecture, we have to check whether the component fits the hard and soft requirements of a Fog Component. Table 6.1 lists the eight properties for a Fog Component. As shown, the `Occupant:Smartphone` fulfilled all hard requirements, but did not offer any of its computational capabilities as a service in the smart environment setup of ARControl. Nevertheless, it can be considered a Fog Component as this is true for any Edge Device according to the Edge Layer definition (Section 4.1). This already provides a first hint on which layer the `Occupant:Smartphone` is placed.

The second requirement for adding a new component to a Fog Architecture is to check whether a bidirectional connection can be established between the component and any existing Fog Component of the Fog Architecture. Thus, adding the component to the Fog Horizon of the Fog Component, and therefore transitively to the Fog Set of the Fog Architecture. In the ARControl setup, in addition to the `Ethernet` connections to the sensors, actuators, and the `OpenHABServer`, every Fog Component on the Fog Layer established a `WIFI` network for Fog Components to connect to. This functionality was supported by Raspberry Pis and enabled a connection of the smartphone to any of the Fog Nodes.

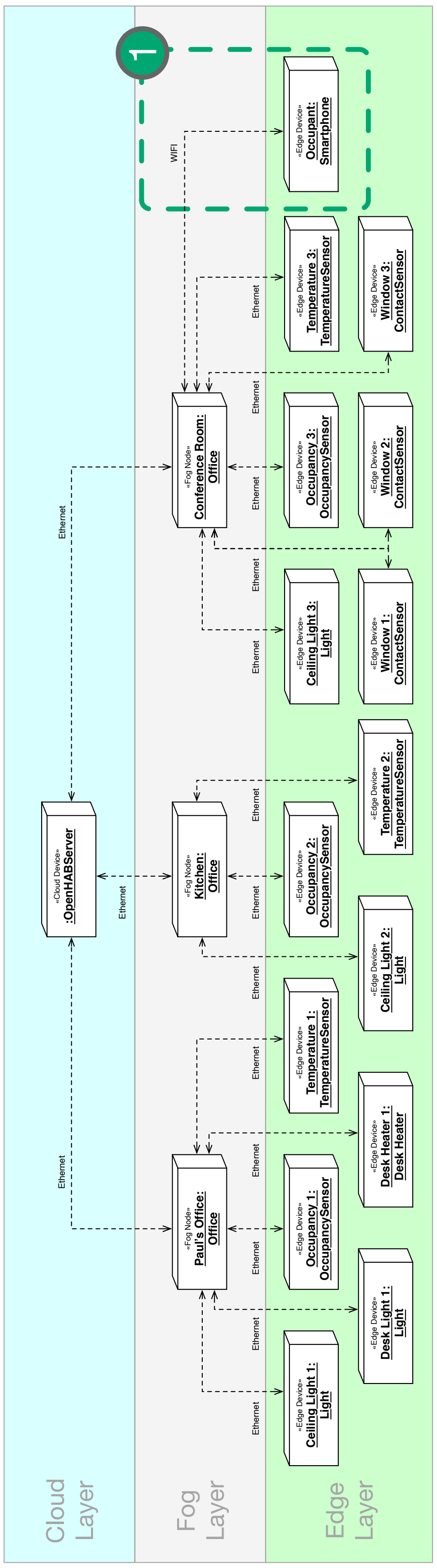

Cloud Layer
Fog Layer
Edge Layer
«Cloud Device» :OpenHABServer
«Fog Node» Paul's Office: Office
«Fog Node» Kitchen: Office
«Fog Node» Conference Room: Office
«Edge Device» Ceiling Light 1: Light
«Edge Device» Desk Light 1: Light
«Edge Device» Occupancy 1: OccupancySensor
«Edge Device» Desk Heater 1: Desk Heater
«Edge Device» Temperature 1: TemperatureSensor
«Edge Device» Ceiling Light 2: Light
«Edge Device» Occupancy 2: OccupancySensor
«Edge Device» Temperature 2: TemperatureSensor
«Edge Device» Ceiling Light 3: Light
«Edge Device» Window 1: ContactSensor
«Edge Device» Occupancy 3: OccupancySensor
«Edge Device» Window 2: ContactSensor
«Edge Device» Temperature 3: TemperatureSensor
«Edge Device» Window 3: ContactSensor
«Edge Device» Occupant: Smartphone
Ethernet
WIFI
1

Figure 6.15: The ARControl setup as shown in Figure 6.14. Additionally, a dynamic Fog Component `Occupant : Smartphone`, which is highlighted by ①, is added to the Fog Architecture.

As shown in Equation 38, this allows us to add the `Occupant:Smartphone` as a new Fog Component to the existing sets. As described in Section 6.1.1, the use case for the smartphone in the ARControl case study was to connect the physical and virtual world by creating an AR scene on top of the camera footage to provide information and enable controls. Therefore, the `Occupant:Smartphone` used the services of the Fog Layer and was placed on the Edge Layer.

Equation 38: ARControl: New Fog Component

$$\mathsf{FogSet}_{new} = \mathsf{FogSet}_{old} \cup \{\mathsf{Occupant}\}$$
$$\mathsf{EdgeLayer}_{new} = \mathsf{EdgeLayer}_{old} \cup \{\mathsf{Occupant}\}$$

To update the Communication Set, we have to determine which Fog Node the `Occupant:Smartphone` connected to. Figure 6.16 shows the three Fog Nodes physically placed in the Intelligent Workplace. The Fog Visibility of `Paul's Office` is shown as a *red* circle, the Fog Visibility of the `Kitchen` is the *green* circle, and the `Conference Room`'s Fog Visibility is represented as the *blue* circle. For readability, we do not show the `Occupant:Smartphone`'s Fog Visibility but imply an overlap with the Fog Nodes. The dashed line from `Paul's Office` through the `Kitchen` into the `Conference Room` represented the path of the occupant. ①, ②, and ③ highlight the positions at which the `Occupant:Smartphone` connected to the different Fog Nodes. At each of those positions, the Communication Set was updated as shown in Equation 39.

Equation 39: ARControl: IWPath

$$\begin{aligned}
① : \mathsf{CommunicationSet}_{new} &= \mathsf{CommunicationSet}_{old} \cup \\
&\quad \{(\mathsf{Paul'sOffice}, \mathsf{WIFI}, \mathsf{Occupant})^{\leftrightarrow}\} \\
② : \mathsf{CommunicationSet}_{new} &= \mathsf{CommunicationSet}_{old} \setminus \\
&\quad \{(\mathsf{Paul'sOffice}, \mathsf{WIFI}, \mathsf{Occupant})^{\leftrightarrow}\} \\
&\quad \cup \{(\mathsf{Kitchen}, \mathsf{WIFI}, \mathsf{Occupant})^{\leftrightarrow}\} \\
③ : \mathsf{CommunicationSet}_{new} &= \mathsf{CommunicationSet}_{old} \setminus \\
&\quad \{(\mathsf{Kitchen}, \mathsf{WIFI}, \mathsf{Occupant})^{\leftrightarrow}\} \\
&\quad \cup \{(\mathsf{ConferenceRoom}, \mathsf{WIFI}, \mathsf{Occupant})^{\leftrightarrow}\}
\end{aligned}$$

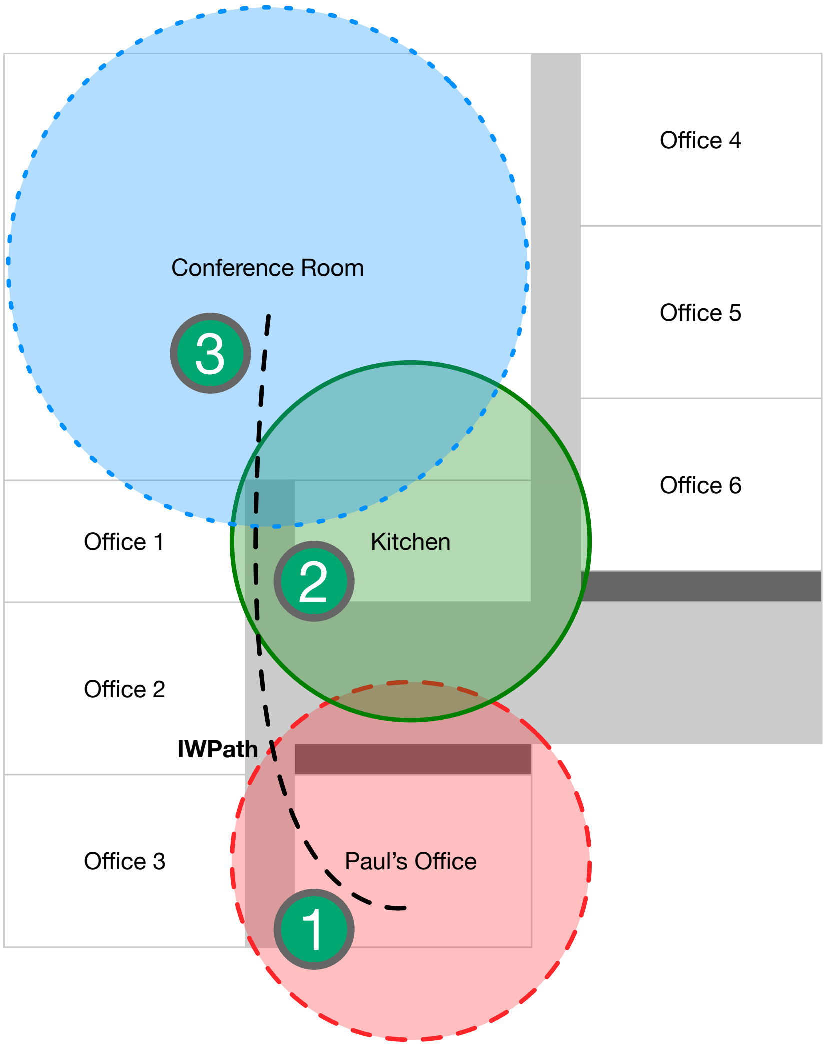


Figure 6.16: Overview of the Fog Node placement in the Intelligent Workplace. The three different circles indicate the Fog Visibilities of the three Fog Nodes: *Paul's Office*, *Kitchen*, and *Conference Room*. The dashed line shows the path of an occupant, and thus the *Occupant:Smartphone*'s position. ①, ②, and ③ highlight position at which the Communication Set is updated.

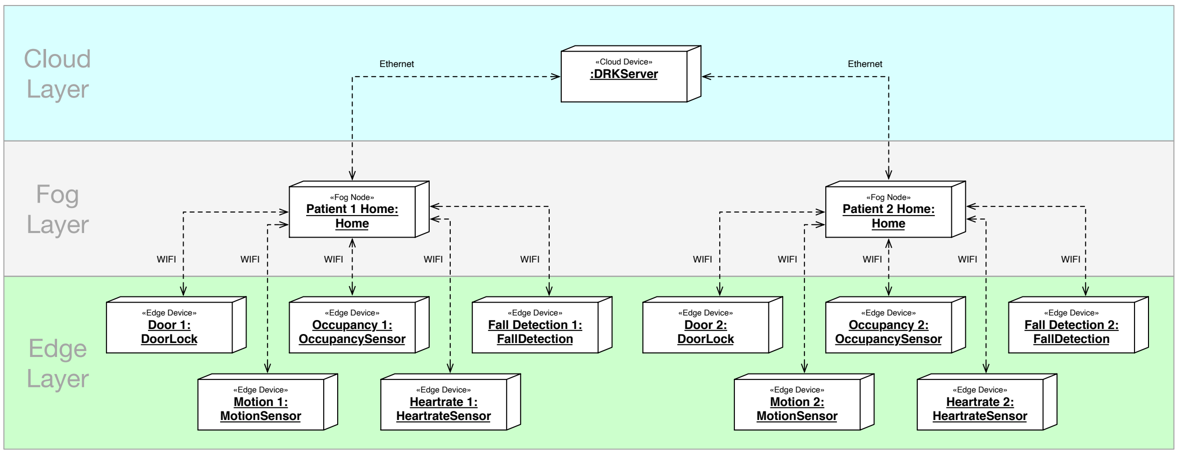

Cloud Layer
Fog Layer
Edge Layer
Ethernet
«Cloud Device» :DRKServer
Ethernet
«Fog Node» Patient 1 Home: Home
«Fog Node» Patient 2 Home: Home
WIFI
«Edge Device» Door 1: DoorLock
«Edge Device» Occupancy 1: OccupancySensor
«Edge Device» Fall Detection 1: FallDetection
«Edge Device» Motion 1: MotionSensor
«Edge Device» Heartrate 1: HeartrateSensor
«Edge Device» Door 2: DoorLock
«Edge Device» Occupancy 2: OccupancySensor
«Edge Device» Fall Detection 2: FallDetection
«Edge Device» Motion 2: MotionSensor
«Edge Device» Heartrate 2: HeartrateSensor

Figure 6.17: The HW/SW mapping of the Lassie setup (UML Deployment Diagram). For readability, we only show components and the used communication channels. Lassie is a 3-layered Fog Architecture containing a *Cloud Layer*, a *Fog Layer*, and *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers.

Lassie

The second case study within the dynamic Fog Components validation is called *Lassie*. As described in Section 6.1.2, Lassie is located in the smart environment and health domain and as such has a few similarities to the first case study. Lassie used a 3-layered Fog Architecture setup which is shown in Figure 6.17.

On the *Cloud Layer*, the `DRKServer` was the central instance from the **D**eutsches **R**otes **K**reuz (German Red Cross). This server stored and evaluated data from different elderly persons, referred to as "patient" in the following. The stored data was used to find anomalies and critical events in the patients daily life. On the one side, rules defined specific events such as "Patient did not leave the bedroom until noon" or "Patient did not open the fridge to get breakfast" which could be accomplished by using motion sensors and occupancy sensors, respectively. On the other side, the data was used to train a personalized machine learning model which tried to learn the daily behavior of the patient, and thus detect unusual behavior of the patient on a daily basis. Additionally, in case of an emergency, a webpage on this server was used to deploy an emergency worker to the patient.

The *Fog Layer* contained the unit within the patient's home which established the connection to the `DRKServer`. This Fog Node aggregated data, and therefore optimized the connection to the `DRKServer`, but also evaluated simple rules. Additionally, it provided WIFI access to which patients connect their smartphones which in turn served as a sensory device or as an interaction device similar to the ARControl application. For simplicity, the Fog Nodes were IoT-Gateways[25] from Q-loud[26].

[25]https://www.q-loud.de/hardware-katalog/iot-gateway
[26]https://www.q-loud.de

Also they did not provide a WIFI access point or the computational power of a Raspberry Pi[27], as most sensors were from the same hardware provider, it simplified the setup of the system in the patient's home. The Lassie setup was described for two homes: `Patient 1 Home` and `Patient 2 Home`.

The Edge Layer involved all the sensors used to detect the patients behavior. All devices on this layer were connected to the Fog Layer using WIFI. Motion sensors were used to detect movement patterns of the patient and provided valuable hints to the patients condition also not always being an emergency. Occupancy sensors provided an approximation of the patients location and allowed conclusions on patients leaving their homes. Heart rate sensors and fall detection devices provided data closer linked to individual patients. Those devices were for instance smart watches or fitness trackers. Meanwhile, smart locks, while being able to provide information about opened and closed doors, were mainly used to grant emergency workers access to the patient's home in case of an emergency. Lassie provided the possibility to include additional sensors and add more patients' homes.

All Fog Components within this setup are included in the Fog Set of the Fog Architecture shown in Equation 40.

Equation 40: Lassie: Fog Set

$$\begin{aligned} FogSet = \{ & DRKServer, Patient1Home, Patient2Home, Door1, \\ & Motion1, Occupancy1, Heartrate1, FallDetection1, \\ & Door2, Motion2, Occupancy2, Heartrate2, FallDetection2\} \end{aligned}$$

Equation 41 shows the Communication Set describing the different communication channels used between the Fog Components. For readability, same as in Section 6.2.1, we do not list every single Fog Component on the Edge Layer as all of those Edge Devices were connected via WIFI, but rather simplify it with one connection to a pseudo Edge Device. Additionally, we use the abbreviated form for symmetrical connections. The entire Communication Set with every Fog Component can be found in the appendix in Equation 88.

Equation 41: Lassie: Communication Set

$$\begin{aligned} CommunicationSet = \{ & (DRKServer, Ethernet, Patient1Home)^{\leftrightarrow}, \\ & (DRKServer, Ethernet, Patient2Home)^{\leftrightarrow}, \\ & (Patient1Home, WIFI, EdgeDevice)^{\leftrightarrow}, \\ & (Patient2Home, WIFI, EdgeDevice)^{\leftrightarrow}\} \end{aligned}$$

[27]https://www.raspberrypi.org

Equation 42 shows the Service Set for Lassie. Same as for ARControl, it contains the same six services: `getGlobalDeviceList`, `getGlobalDeviceValue`, `setGlobalDeviceValue`, `getDeviceList`, `getDeviceValue`, and `setDeviceValue`. This is based on their similar domain which both focus on the collection and evaluation of sensor data. Thus, the services were used for the same use cases. In comparison to the smart environment in which the cloud allowed occupants to control the smart environment even when they ere not in the smart environment, in Lassie, the cloud service `setGlobalDeviceValue` allowed emergency workers to open the doors of patients in case of an emergency and request additional data of the patient using `getGlobalDeviceValue`.

Equation 42: Lassie: Service Set

$$\begin{aligned}
ServiceSet = \{&getGlobalDeviceList, getGlobalDeviceValue,\\
&setGlobalDeviceValue, getDeviceList,\\
&getDeviceValue, setDeviceValue\}\\
ServiceSetMapping = \{(&\{\}, CloudLayer, \{getGlobalDeviceList,\\
&\quad getGlobalDeviceValue, setGlobalDeviceValue\}),\\
(&\{getGlobalDeviceList, getGlobalDeviceValue,\\
&\quad setGlobalDeviceValue\}, FogLayer,\\
&\{getDeviceList, getDeviceValue, setDeviceValue\}),\\
(&\{getDeviceList, getDeviceValue, setDeviceValue\},\\
&EdgeLayer, \{\})\}
\end{aligned}$$

The according layers are shown in Equation 43. All "global" services were offered by the cloud, alias the `DRKServer` as indicated by the *Service Set Mapping*. Those services were used by the Fog Nodes which in turn offered related services to the sensors for collecting data and to the patient's smartphone to control actuators such as `Door 1`.

Equation 43: Lassie: Layer Definitions

$$\begin{aligned}
CloudLayer &= \{DRKServer\}\\
FogLayer &= \{Patient1Home, Patient2Home\}\\
EdgeLayer &= \{Door1, Motion1, Occupancy1, Heartrate1, FallDetection1,\\
&\quad Door2, Motion2, Occupancy2, Heartrate2, FallDetection2\}
\end{aligned}$$

The dynamically added and removed component is shown in Figure 6.18 with ① which represented the `Patient 2:Smartphone`.

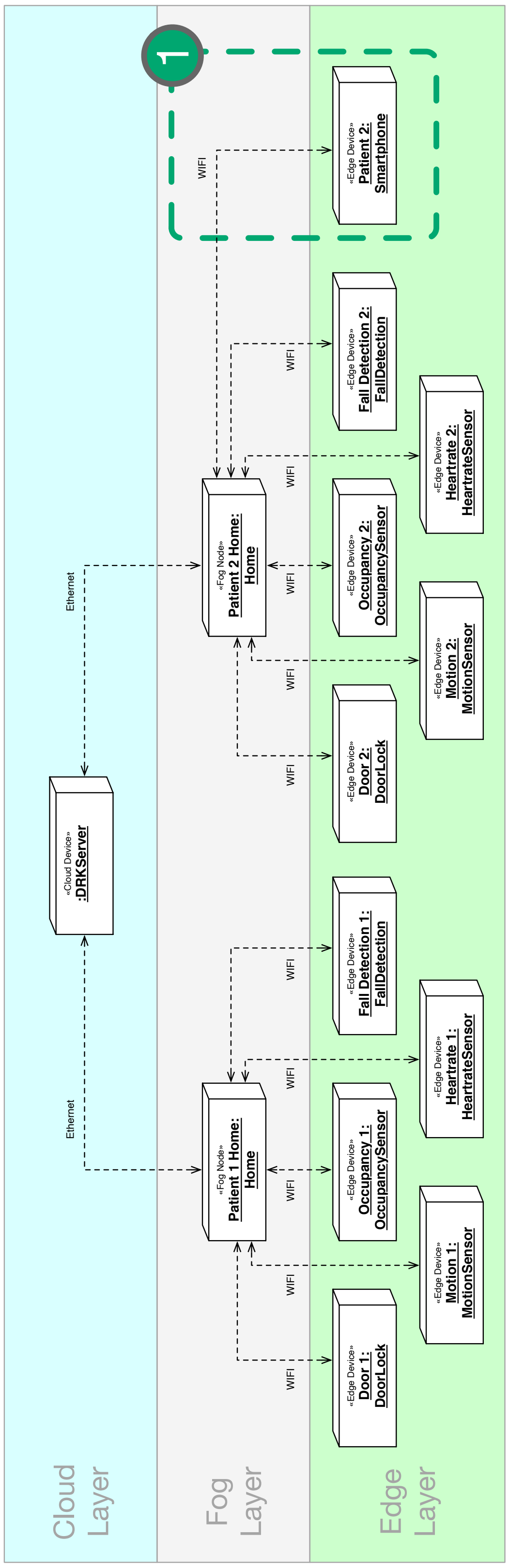

Cloud Layer
Fog Layer
Edge Layer
«Cloud Device» :DRKServer
Ethernet
Ethernet
«Fog Node» Patient 1 Home: Home
«Fog Node» Patient 2 Home: Home
WIFI
«Edge Device» Door 1: DoorLock
«Edge Device» Motion 1: MotionSensor
«Edge Device» Occupancy 1: OccupancySensor
«Edge Device» Heartrate 1: HeartrateSensor
«Edge Device» Fall Detection 1: FallDetection
«Edge Device» Door 2: DoorLock
«Edge Device» Motion 2: MotionSensor
«Edge Device» Occupancy 2: OccupancySensor
«Edge Device» Heartrate 2: HeartrateSensor
«Edge Device» Fall Detection 2: FallDetection
«Edge Device» Patient 2: Smartphone
1

Figure 6.18: The Lassie setup as shown in Figure 6.17. ① highlights the *Patient 2:Smartphone* which dynamically moves in and out of the Fog Architecture according to the patient's location.

Table 6.2: Hard and soft requirements of Fog Components according to Section 4.1 for the dynamic Fog Component: `Patient 2:Smartphone`. Hard requirements are highlighted in *red*, soft requirements in *orange*.

		`Patient 2:Smartphone`
1.	Interconnectivity	✓
2.	Information Sharing	✓
3.	Uniquely Addressable	✓
4.	Computational Capabilities	✓
5.	Wireless communication	✓
6.	Locality	✓
7.	General-purpose Computational Capabilities	✓
8.	Offers Capabilities as Service	X

First, we check if the component complies to the hard and soft requirements of a Fog Component as shown in Table 6.2. As for every smartphone, the hard requirements for a Fog Component were met. But, depending on the setup, the soft requirements might change. For example, if we would have used a smart watch as a heart rate or fall detection device that relied on a connection to the smartphone to provide the data to the patient's home Fog Node, the smartphone would have offered services, and thus *Offer[s] Capabilities as Service*. As this was not the case in our setup, it is the only requirement which is not checked.

After ensuring that the `Patient 2:Smartphone` was a Fog Component, it can be added to the Fog Set as shown in Equation 44. Additionally, as already described by the role that the smartphone plays in the Lassie setup, it was added to the Edge Layer.

Equation 44: Lassie: New Fog Component

$$\begin{aligned} \mathsf{FogSet}_{new} &= \mathsf{FogSet}_{old} \cup \{\mathsf{Patient2} : \mathsf{Smartphone}\} \\ \mathsf{EdgeLayer}_{new} &= \mathsf{EdgeLayer}_{old} \cup \{\mathsf{Patient2} : \mathsf{Smartphone}\} \end{aligned}$$

The Communication Set needs to be updated as shown in Equation 45 whenever patients entered or left their home.

Equation 45: Lassie: Entering or Leaving Home

$$\begin{aligned} \mathsf{CommunicationSet}_{new} &= \mathsf{CommunicationSet}_{old} \cup \\ &\quad \{(\mathsf{Patient2Home}, \mathsf{WIFI}, \mathsf{Patient2} : \mathsf{Smartphone})^{\leftrightarrow}\} \\ \mathsf{CommunicationSet}_{new} &= \mathsf{CommunicationSet}_{old} \setminus \\ &\quad \{(\mathsf{Patient2Home}, \mathsf{WIFI}, \mathsf{Patient2} : \mathsf{Smartphone})^{\leftrightarrow}\} \end{aligned}$$

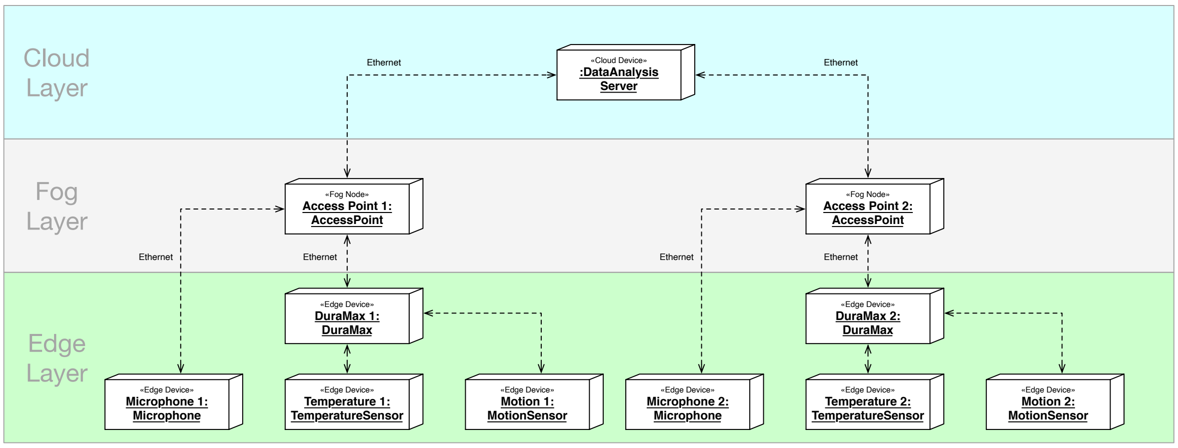

Cloud Layer
Fog Layer
Edge Layer
Ethernet
«Cloud Device» :DataAnalysis Server
«Fog Node» Access Point 1: AccessPoint
«Fog Node» Access Point 2: AccessPoint
«Edge Device» DuraMax 1: DuraMax
«Edge Device» DuraMax 2: DuraMax
«Edge Device» Microphone 1: Microphone
«Edge Device» Temperature 1: TemperatureSensor
«Edge Device» Motion 1: MotionSensor
«Edge Device» Microphone 2: Microphone
«Edge Device» Temperature 2: TemperatureSensor
«Edge Device» Motion 2: MotionSensor

Figure 6.19: The HW/SW mapping of the PdMFrame setup (UML Deployment Diagram). For readability, we only show components and the used communication channels. PdMFrame is a 3-layered Fog Architecture containing a *Cloud Layer*, a *Fog Layer*, and *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers.

PdMFrame

The third case study for the dynamic Fog Component validation is *PdMFrame* introduced in Section 6.1.3. The setup, as shown in Figure 6.19, was a 3-layered Fog Architecture and accordingly, a 3-tiered Fog Architecture.

The *Cloud Layer* contained the `Data Analysis Server` which was the central server instance of the predictive maintenance framework. It was used to store sensor data, store a variety of machine learning algorithms, and train machine learning models to predict machine failures. Those machine learning models were compared based on the data provided by the sensors. The best performing model was selected and provided to be used for future incoming sensor data. Based on the users' feedback, the models could constantly be retrained to incorporate new data based on the process shown and described in Section 6.1.3.

The *Fog Layer* contained the access points to the machine under test. These access points provided the connection to the *Cloud Layer* and aggregated sensor data from the machine as well as external sensors. Additionally, they performed a preprocessing of the sensor data to utilize the computational power of the access point. The sensor data was transformed from the format that the sensors provide to the format that was expected by the `Data Analysis Server`. In the setup at hand, the access point was represented by a laptop which was directly connected to the machine under test and the external sensors, in our case a directed microphone as shown in Figure 6.8.

The *Edge Layer* contained the sensors and the machines under test. The tested machines were DuraMax[28] machines by ZEISS[29]. Those machines use a coordinate systems and a VAST XXT scanning sensor[30] to check manufactured parts against their specification. While doing so, the scanning sensor is moved by three motors coordinating each axis. Among other sounds, the sound of these moving axis was recorded by a directed microphone and used to predict machine failures. Within the DuraMax machines, a temperature sensor and a motion sensor monitored the internal machines behavior. Also they are displayed separately in Figure 6.19 to provide an overview of available sensors, they were build in the machine and could only be accessed via the machine itself. Thus, we consider the DuraMax, temperature sensor, and motion sensor as one Fog Component on the *Edge Layer*.

All Fog Components of this Fog Architecture are contained in the Fog Set which is shown in Equation 46.

Equation 46: PdMFrame: Fog Set

$$\begin{aligned} FogSet = \{&DataAnalysisServer, AccessPoint1, AccessPoint2,\\ &DuraMax1, Microphone1, Temperature1, Motion1,\\ &DuraMax2, Microphone2, Temperature2, Motion2\} \end{aligned}$$

Equation 47 presents the Communication Set for PdMFrame. We use the same simplification as for ARControl and Lassie: The reflexive indicator means that the connection is bidirectional and the Edge Devices are combined in a single pseudo Edge Device. As all communication channels are ethernet, we do not lose any information by doing so. For the sake of completeness, we included the entire Communication Set for PdMFrame in the appendix in Section B.3.

Equation 47: PdMFrame: Communication Set

$$\begin{aligned} CommunicationSet = \{&(DataAnalysisServer, Ethernet, AccessPoint1)^{\leftrightarrow},\\ &(DataAnalysisServer, Ethernet, AccessPoint2)^{\leftrightarrow},\\ &(AccessPoint1, Ethernet, EdgeDevices)^{\leftrightarrow},\\ &(AccessPoint2, Ethernet, EdgeDevices)^{\leftrightarrow}\} \end{aligned}$$

[28]https://www.zeiss.de/messtechnik/produkte/systeme/koordinatenmessgeraete/fertigungsmessgeraete/duramax.html

[29]https://www.zeiss.de

[30]https://www.zeiss.de/messtechnik/produkte/sensoren/am-kmg/taktile-sensoren/vast-xxt.html

The Service Set, as shown in Equation 48, consisted of three services: `globalDataAccessPoint`, `retrainModel`, and `dataAccessPoint`. These services were mapped to the different layers according to the Service Set Mapping: The `dataAccessPoint` was the service provided by the `Access Points`. It offered an endpoint to send sensor data, which was aggregated and preprocessed afterwards. The according `globalDataAccessPoint` was the data endpoint of the Cloud Layer. It accepted the preprocessed data and evaluated, persisted, or created a machine learning model for predictive maintenance. The last service `retrainModel` could be used to retrain the stored machine learning model with the data that was stored until the given point in time.

Equation 48: PdMFrame: Service Set

$$\begin{aligned}
ServiceSet &= \{globalDataAccessPoint, retrainModel, dataAccessPoint\} \\
ServiceSetMapping &= \{(\{\}, CloudLayer, \\
&\quad \{globalDataAccessPoint, retrainModel\}), \\
&\quad (\{globalDataAccessPoint, retrainModel\}, \\
&\quad FogLayer, \{dataAccessPoint\}), \\
&\quad (\{dataAccessPoint\}, EdgeLayer, \{\})\}
\end{aligned}$$

According to the requested and offered services, Equation 49 shows the Fog Components' affiliation to the different layers. The `Data Analysis Server` offering the two services `globalDataAccessPoint` and `retrainModel` did not use any service itself and was therefore placed on the *Cloud Layer*. The access points used the `globalDataAccessPoint` and provided a local `dataAccessPoint` service for the locally connected machines und sensors. They were on the Fog Layer. Finally, on the Edge Layer, we placed all the sensors that were used to track the status of the machines under test which used this local `dataAccessPoint` but did not offer any services themselves.

Equation 49: PdMFrame: Layer Definitions

$$\begin{aligned}
CloudLayer &= \{DataAnalysisServer\} \\
FogLayer &= \{AccessPoint1, AccessPoint2\} \\
EdgeLayer &= \{DuraMax1, Microphone1, Temperature1, Motion1, \\
&\quad DuraMax2, Microphone2, Temperature2, Motion2\}
\end{aligned}$$

The component that we wanted to add to the Fog Architecture is shown in Figure 6.20 highlighted by ①. It represented a new `Access Point 3` with another `DuraMax 3` machine and the corresponding sensors connected to it.

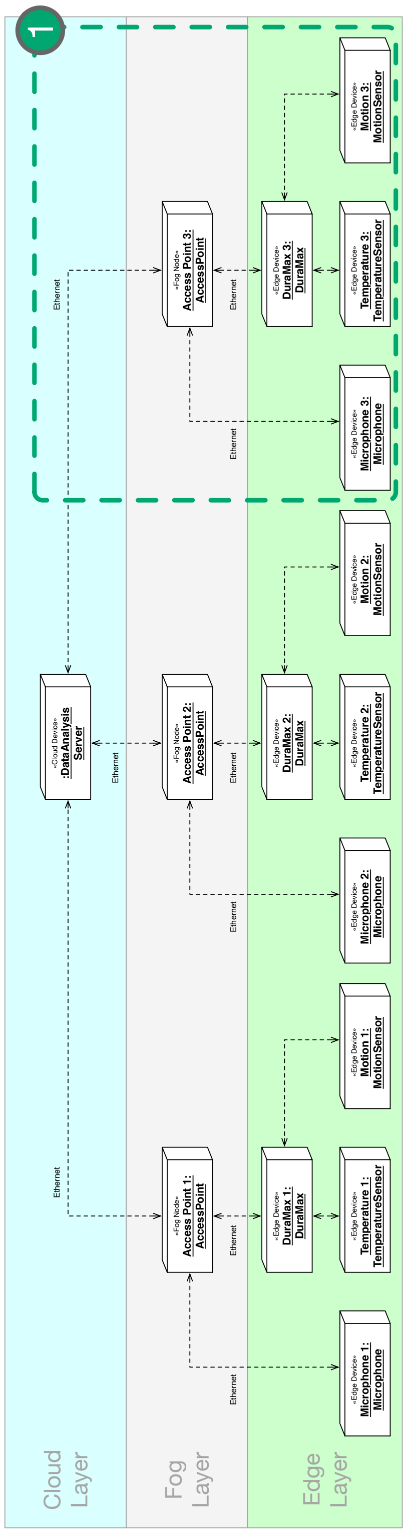

Cloud Layer
Fog Layer
Edge Layer
1
«Cloud Device» :DataAnalysis Server
«Fog Node» Access Point 1: AccessPoint
«Fog Node» Access Point 2: AccessPoint
«Fog Node» Access Point 3: AccessPoint
«Edge Device» DuraMax 1: DuraMax
«Edge Device» DuraMax 2: DuraMax
«Edge Device» DuraMax 3: DuraMax
«Edge Device» Microphone 1: Microphone
«Edge Device» Microphone 2: Microphone
«Edge Device» Microphone 3: Microphone
«Edge Device» Temperature 1: TemperatureSensor
«Edge Device» Temperature 2: TemperatureSensor
«Edge Device» Temperature 3: TemperatureSensor
«Edge Device» Motion 1: MotionSensor
«Edge Device» Motion 2: MotionSensor
«Edge Device» Motion 3: MotionSensor
Ethernet

Figure 6.20: The PdMFrame setup as shown in Figure 6.19. ① highlights an additional `Access Point 3` which is dynamically added to the Fog Architecture. Using transitive connections, it adds four Edge Devices.

Table 6.3: Hard and soft requirements of Fog Components according to Section 4.1 for the added Fog Component: `Access Point 3`. Due to the transitive closure of the added component, it also shows the additionally added Edge Devices. Hard requirements are highlighted in *red*, soft requirements in *orange*.

		Access Point 3	DuraMax 3	Microphone 3	Temperature 3	Motion 3
1.	Interconnectivity	✓	✓	✓	✓	✓
2.	Information Sharing	✓	✓	✓	✓	✓
3.	Uniquely Addressable	✓	✓	✓	✓	✓
4.	Computational Capabilities	✓	✓	✓	✓	✓
5.	Wireless communication	X	X	X	X	X
6.	Locality	✓	✓	✓	✓	✓
7.	General-purpose Computational Capabilities	✓	✓	X	X	X
8.	Offers Capabilities as Service	✓	X	X	X	X

As for the case studies before, we first check if the `Access Point 3` conforms to the requirements imposed by the Fog Component definition.
As adding `Access Point 3` transitively adds `Microphone 3`, the `DuraMax 3` machine, `Temperature 3`, and `Motion 3`, we have to check the Fog Component conformance for each of those components as well.

Table 6.3 shows the mapping of the components to the hard and soft requirements. As all hard requirements are checked, the devices comply to the Fog Component definition also they are limited as shown by the soft requirements which are only partly checked. Non of the components integrated wireless communication, most did not have general-purpose computational capabilities, and only the `Access Point 3` provided a service to other components.

After adding the `Access Point 3` to the Fog Set and the *Fog Layer*, as shown in Equation 50, we have to recursively evaluate which other Fog Components will be transitively included in the Fog Architecture. Thus, we added the `DuraMax 3`, the `Microphone 3`, the `Temperature 3`, and the `Motion 3` to the Fog Set (FogSet_{new2}). Additionally, we had to add those Fog Components on their respective layers, in this case the *Edge Layer*.

Equation 50: PdMFrame: New Fog Component

$$\begin{aligned}
\mathsf{FogSet}_{new} &= \mathsf{FogSet}_{old} \cup \{\mathsf{AccessPoint3}\}\\
\mathsf{FogLayer}_{new} &= \mathsf{FogLayer}_{old} \cup \{\mathsf{AccessPoint3}\}\\
\\
\mathsf{FogSet}_{new2} &= \mathsf{FogSet}_{new} \cup \{\mathsf{DuraMax3}, \mathsf{Microphone3},\\
&\quad \mathsf{Temperature3}, \mathsf{Motion3}\}\\
\mathsf{EdgeLayer}_{new} &= \mathsf{EdgeLayer}_{old} \cup \{\mathsf{DuraMax3}, \mathsf{Microphone3},\\
&\quad \mathsf{Temperature3}, \mathsf{Motion3}\}
\end{aligned}$$

For the Communication Set, as shown in Equation 51, we do the same.
First, we add the Communication Component for the connection between the `Data Analysis Server` and the `Access Point 3`. Second, we add the transitive Communication Components for the machine and the sensors ($\mathsf{CommunicationSet}_{new2}$).

Equation 51: PdMFrame: Adding a new Access Point

$$\begin{aligned}
\mathsf{CommunicationSet}_{new} &= \mathsf{CommunicationSet}_{old} \cup\\
&\quad \{(\mathsf{DataAnalysisServer}, \mathsf{Ethernet}, \mathsf{AccessPoint3})^{\leftrightarrow}\}\\
\mathsf{CommunicationSet}_{new2} &= \mathsf{CommunicationSet}_{new} \cup\\
&\quad \{(\mathsf{AccessPoint3}, \mathsf{Ethernet}, \mathsf{Microphone3})^{\leftrightarrow},\\
&\quad (\mathsf{AccessPoint3}, \mathsf{Ethernet}, \mathsf{DuraMax3})^{\leftrightarrow},\\
&\quad (\mathsf{AccessPoint3}, \mathsf{Ethernet}, \mathsf{Temperature3})^{\leftrightarrow},\\
&\quad (\mathsf{AccessPoint3}, \mathsf{Ethernet}, \mathsf{Motion3})^{\leftrightarrow}\}
\end{aligned}$$

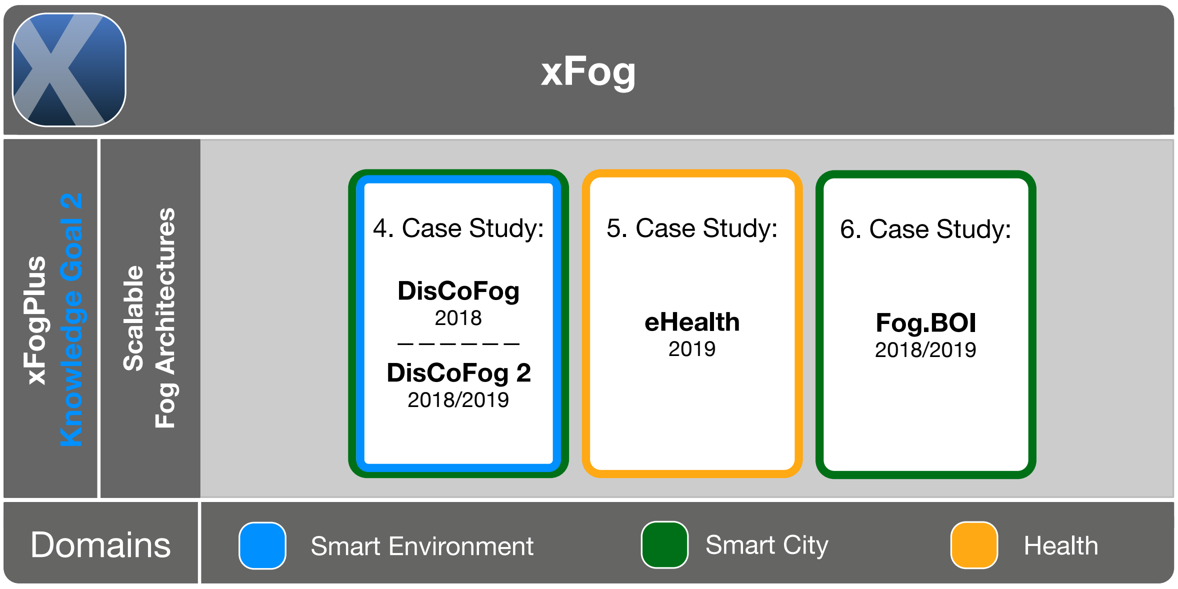


Figure 6.21: Excerpt of the validation design overview shown in Figure 6.1 for Scalable Fog Architectures.

6.2.2 Scalable Fog Architectures

The results of the second validation address the concept of scalable Fog Architectures. We use the three case studies as shown in Figure 6.21: *DisCoFog*, which consists of *DisCoFog 1* and *DisCoFog 2*, *eHealth*, and *Fog.BOI*.

These include the scalable aspects of Fog Architectures, and thus the addition of multiple layers. First, we present an overview of the architecture using the same simplified HW/SW mapping already used for the dynamic Fog Component validation. We use this model to describe the sets that define the Fog Architecture and which need to change in order to add new layers. Second, we set different Abstraction Levels to establish Views on the Fog Architecture and present the resulting point of focus. Finally, we highlight some of the resulting dynamic type changes that occur due to the View concept.

DisCoFog

The first case study in the validation of scalable Fog Architectures is *DisCoFog*. DisCoFog is split into two implementations: *DisCoFog 1* and *DisCoFog 2*. As already described in Section 6.1.4, they differ in the application domain with DisCoFog 1 being placed in a smart city domain using drones and DisCoFog 2 being in the smart environment domain similar to ARControl (Section 6.1.1).

DisCoFog 1: As shown in Figure 6.22, the initial Fog Architecture that we established for DisCoFog 1 consisted of three layers which also related to tiers: the *Cloud Layer*, a single *Fog Layer*, and the *Edge Layer*.

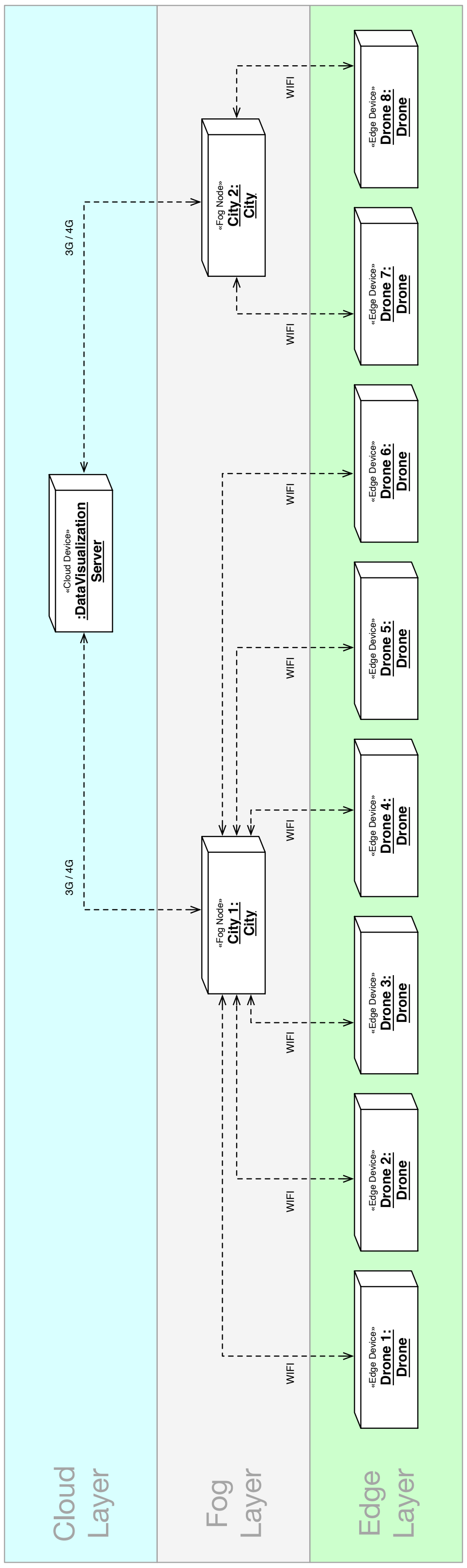


Figure 6.22: The HW/SW mapping of the DisCoFog setup (UML Deployment Diagram). For readability, we only show components and the used communication channels. DisCoFog is a 3-layered Fog Architecture containing a *Cloud Layer*, a *Fog Layer*, and *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers.

On the *Cloud Layer*, we had one single Fog Component, the `Data Visualization Server`. It was used to render three-dimensional heat maps from the movement of drones which is a computational heavy task. Therefore, it collected drone movement data.

The movement data was gathered on the *Fog Layer* which contained the Fog Nodes: `City 1` and `City 2`. They were used to provide local access points for the Edge Devices to connect using `WIFI`, a local area network. The locality was used to provide fast updates for other drones to avoid collisions within the cities' reach. Additionally, they established a connection to the `Data Visualization Server` to post the drone movements of their area. In the test setup, the Fog Nodes were represented by smartphones that offered hotspots for the drones to connect while maintaining their connection to the cloud using `3G / 4G`.

Finally, on the *Edge Layer*, we placed the drones. To show the practicality of additional layers, we modeled the system for a total of eight drones.

Equation 52 shows the Fog Architecture set which contains the three layers. According to the definition provided in Section 4.2, its cardinality is three with only a single Fog Layer in the Fog Layer Set.

Equation 52: DisCoFog 1: Fog Architecture

$$\mathsf{FogArchitecture} := \{\mathsf{EdgeLayer},\ \mathsf{FogLayer},\ \mathsf{CloudLayer}\}$$
$$|\mathsf{FogArchitecture}| = 2 + |\mathsf{FogLayerSet}| = 3$$

These layers contained Fog Components which comprise the Fog Set. As the dynamic addition of Fog Components was not the focus in this validation, we show the Fog Set and the layers before and after adding the new layer in appendix Section B.4.

The Service Set of the initial Fog Architecture is shown in Equation 53 with the mapping between the services and the layers which consumed and offered them. In total the Fog Architecture contained three services: `uploadMovementData`, `getDronePositions`, and `uploadGlobalMovementData`. To describe the mapping between the offered and consumed services, we use a triple with the three parts:

$$(\{\mathsf{ConsumedServices}\},\ \mathsf{Layer},\ \{\mathsf{ProvidedServices}\})$$

Thus, it is a mapping from two sets of services on a layer, which is a set of Fog Components.

Equation 53: DisCoFog 1: Service Set and Mapping

$$\begin{aligned}
&\mathit{ServiceSet} = \{\mathit{uploadMovementData}, \mathit{getDronePositions},\\
&\qquad\qquad \mathit{uploadGlobalMovementData}\}\\
&\mathit{ServiceSetMapping} = \{\\
&\quad (\{\}, \mathit{CloudLayer}, \{\mathit{uploadGlobalMovementData}\}),\\
&\quad (\{\mathit{uploadGlobalMovementData}\}, \mathit{FogLayer},\\
&\quad \{\mathit{uploadMovementData}, \mathit{getDronePositions}\}),\\
&\quad (\{\mathit{uploadMovementData}, \mathit{getDronePositions}\}, \mathit{EdgeLayer}, \{\})\\
&\}
\end{aligned}$$

When adding a new layer to a Fog Architecture, several Communication Components have to be removed and newly added to the Communication Set. Therefore, Equation 54 describes the Communication Set of the initial Fog Architecture. For readability, we stick to the abbreviated representation introduced in the dynamic Fog Component validation (Section 6.2.1).

Equation 54: DisCoFog 1: Communication Set

$$\begin{aligned}
\mathit{CommunicationSet} = \{&(\mathit{DataVisualizationServer}, 3G/4G, \mathit{City1})^{\leftrightarrow},\\
&(\mathit{DataVisualizationServer}, 3G/4G, \mathit{City2})^{\leftrightarrow},\\
&(\mathit{City1}, \mathit{WIFI}, \mathit{EdgeDevice})^{\leftrightarrow},\\
&(\mathit{City2}, \mathit{WIFI}, \mathit{EdgeDevice})^{\leftrightarrow}\}
\end{aligned}$$

After introducing the most important sets which are needed to add a new layer to the Fog Architecture, Figure 6.23 shows the Fog Architecture setup including the newly added layer indicated by ①. The new layer contained three new Fog Components: `City 1 Access 1`, `City 1 Access 2`, and `City 2 Access 1`. Each access point spread the communication load of the city Fog Components and extended the area that could be covered.

In accordance to the added layer, we have to adjust the previously described sets to include the Fog Components, services, as well as Communication Components. Equation 55 shows the resulting Fog Architecture definition, the Service Set, the Service Set Mapping, and the Communication Set.

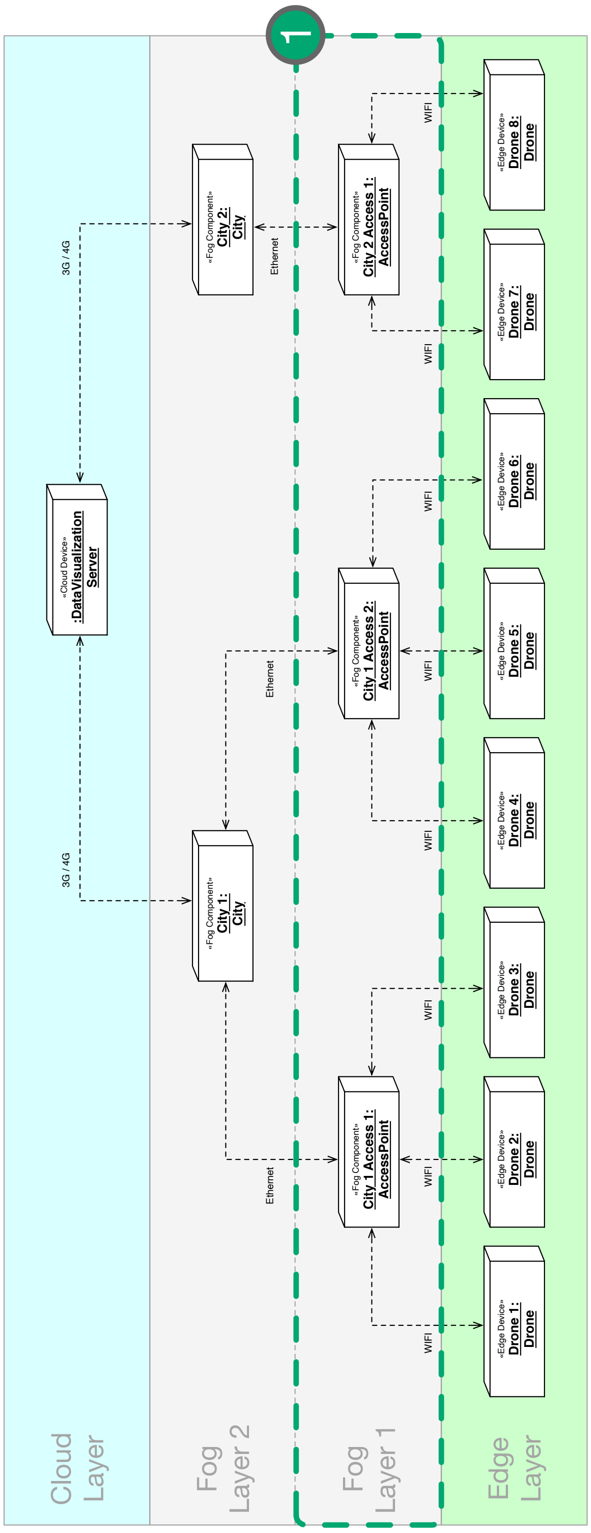

Cloud Layer
Fog Layer 2
Fog Layer 1
Edge Layer
1
«Cloud Device» :DataVisualization Server
3G / 4G
3G / 4G
«Fog Component» City 1: City
«Fog Component» City 2: City
Ethernet
Ethernet
Ethernet
«Fog Component» City 1 Access 1: AccessPoint
«Fog Component» City 1 Access 2: AccessPoint
«Fog Component» City 2 Access 1: AccessPoint
WIFI
«Edge Device» Drone 1: Drone
«Edge Device» Drone 2: Drone
«Edge Device» Drone 3: Drone
«Edge Device» Drone 4: Drone
«Edge Device» Drone 5: Drone
«Edge Device» Drone 6: Drone
«Edge Device» Drone 7: Drone
«Edge Device» Drone 8: Drone

Figure 6.23: The HW/SW mapping of the DisCoFog setup after adding a new layer (UML Deployment Diagram). For readability, we only show components and the used communication channels. After adding the new layer, DisCoFog is a 4-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer2*, *Fog Layer 1*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers. ① highlights the added layer and all the contained Fog Components.

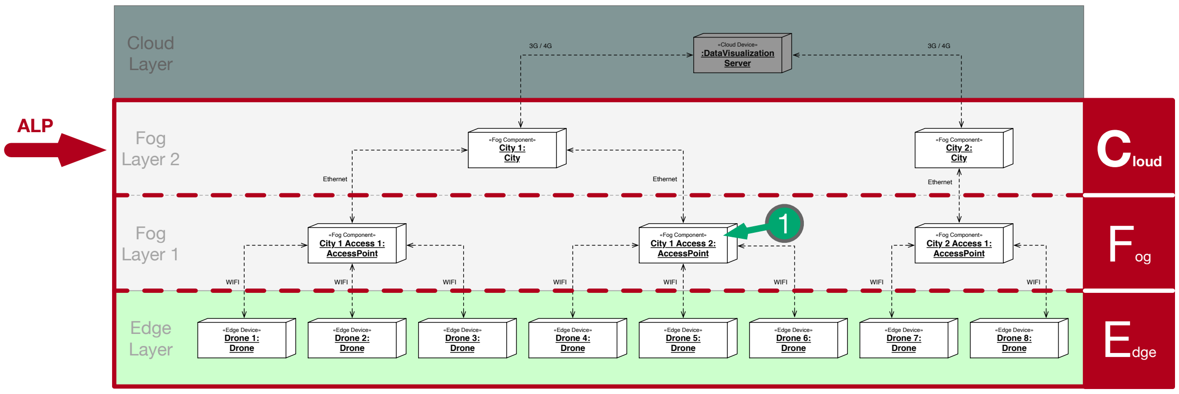
Equation 55: DisCoFog 1: Layer Addition

$$\begin{aligned}
\mathsf{FogArchitecture}_{new} &= \mathsf{FogArchitecture}_{old} \setminus \{\mathsf{FogLayer}\} \cup \\
&\quad \{\mathsf{FogLayer1}, \mathsf{FogLayer2}\} \\
|\mathsf{FogArchitecture}| &= 2 + |\mathsf{FogLayerSet}| = 4 \\
\mathsf{ServiceSet}_{new} &= \mathsf{ServiceSet}_{old} \cup \\
&\quad \{\mathsf{uploadMovementData}^*, \mathsf{getDronePositions}^*\}
\end{aligned}$$

$$\begin{aligned}
&\mathsf{ServiceSetMapping}_{new} = \\
&\quad \mathsf{ServiceSetMapping}_{old} \setminus \{(\{\mathsf{uploadGlobalMovementData}\}, \\
&\quad \mathsf{FogLayer}, \{\mathsf{uploadMovementData}, \mathsf{getDronePositions}\})\} \\
&\quad \cup \\
&\quad \{(\{\mathsf{uploadMovementData}^*, \mathsf{getDronePositions}^*\}, \mathsf{FogLayer1}, \\
&\quad \{\mathsf{uploadMovementData}, \mathsf{getDronePositions}\}), \\
&\quad (\{\mathsf{uploadGlobalMovementData}\}, \mathsf{FogLayer2}, \\
&\quad \{\mathsf{uploadMovementData}^*, \mathsf{getDronePositions}^*\})\}
\end{aligned}$$

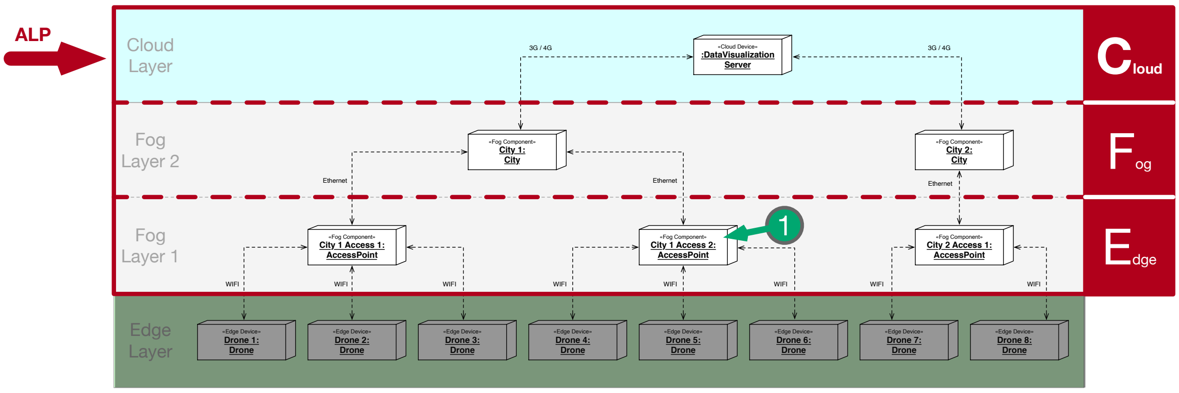
$$\begin{aligned}
\mathsf{CommunicationSet} &= \mathsf{CommunicationSet}_{old} \setminus \\
&\quad \{(\mathsf{City1}, \mathsf{WIFI}, \mathsf{EdgeDevice})^{\leftrightarrow}, \\
&\quad (\mathsf{City2}, \mathsf{WIFI}, \mathsf{EdgeDevice})^{\leftrightarrow}\} \\
&\quad \cup \\
&\quad \{(\mathsf{City1}, \mathsf{WIFI}, \mathsf{City1AccessPoint1})^{\leftrightarrow}, \\
&\quad (\mathsf{City1}, \mathsf{WIFI}, \mathsf{City1Access2})^{\leftrightarrow}, \\
&\quad (\mathsf{City2}, \mathsf{WIFI}, \mathsf{City2Access1})^{\leftrightarrow}, \\
&\quad (\mathsf{City1Access1}, \mathsf{WIFI}, \mathsf{EdgeDevice})^{\leftrightarrow}, \\
&\quad (\mathsf{City1Access2}, \mathsf{WIFI}, \mathsf{EdgeDevice})^{\leftrightarrow}, \\
&\quad (\mathsf{City2Access1}, \mathsf{WIFI}, \mathsf{EdgeDevice})^{\leftrightarrow}\}
\end{aligned}$$

For the Fog Architecture definition, we have to remove the previous Fog Layer which needs to be updated to reflect the number. Thus, the new Fog Layer is called *Fog Layer 1* and the already existing layer is *Fog Layer 2* based on the definition from Section 4.2 which defines to count starting from the lowest layer. In the Service Set, we add the two new services which are offered by *Fog Layer 2* in the new setup: $\mathsf{uploadMovementData}^*$ and $\mathsf{getDronePosition}^*$. The Service Set Mapping shows this new service mapping by assigning the respective services to *Fog Layer 1* and *Fog Layer 2*.



ALP
Cloud Layer
Fog Layer 2
Fog Layer 1
Edge Layer
«Cloud Device» :DataVisualization Server
3G / 4G
3G / 4G
«Fog Component» City 1: City
«Fog Component» City 2: City
Ethernet
«Fog Component» City 1 Access 1: AccessPoint
«Fog Component» City 1 Access 2: AccessPoint
«Fog Component» City 2 Access 1: AccessPoint
WIFI
«Edge Device» Drone 1: Drone
«Edge Device» Drone 2: Drone
«Edge Device» Drone 3: Drone
«Edge Device» Drone 4: Drone
«Edge Device» Drone 5: Drone
«Edge Device» Drone 6: Drone
«Edge Device» Drone 7: Drone
«Edge Device» Drone 8: Drone
1
Cloud
Fog
Edge

(a) View$_1$: The first 3-layered View of the Fog Architecture. The Abstraction Level is set to the third layer which relates to the `Fog Layer 2`.



ALP
Cloud Layer
Fog Layer 2
Fog Layer 1
Edge Layer
«Cloud Device» :DataVisualization Server
3G / 4G
3G / 4G
«Fog Component» City 1: City
«Fog Component» City 2: City
Ethernet
«Fog Component» City 1 Access 1: AccessPoint
«Fog Component» City 1 Access 2: AccessPoint
«Fog Component» City 2 Access 1: AccessPoint
WIFI
«Edge Device» Drone 1: Drone
«Edge Device» Drone 2: Drone
«Edge Device» Drone 3: Drone
«Edge Device» Drone 4: Drone
«Edge Device» Drone 5: Drone
«Edge Device» Drone 6: Drone
«Edge Device» Drone 7: Drone
«Edge Device» Drone 8: Drone
1
Cloud
Fog
Edge

(b) View$_2$The second 3-layered View of the Fog Architecture. The Abstraction Level is set to the fourth layer which relates to the `Cloud Layer`.

Figure 6.24: The HW/SW mapping of the DisCoFog showing the different 3-layered Views (UML Deployment Diagram). For readability, we only show components and the used communication channels. It represents a 4-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer2*, *Fog Layer 1*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers. ① highlights one example Fog Component which dynamically changes its *Fog Type* based on the View it is part of.

Finally, in the Communication Set, we remove the Communication Components that established the connection between the cities and the drones and add the Communication Components for the connection between the cities and their added access points, and the access points and the drones.

With the four layered Fog Architecture, we can differentiate between two 3-layered Views as shown in Figure 6.24. Accordingly, the Abstraction Levels are set to the third layer (*Fog Layer 2*) and the fourth layer (*Cloud Layer*). Based on the View definition introduced in Section 4.2.2, Equation 56 shows the two Views.

Equation 56: DisCoFog: Views

$$View(ALP, n) := \bigcup_{i=0}^{n-1} Layer(ALP - i)$$

$$\begin{aligned} View_1(2,3) :&= \bigcup_{i=0}^{n-1} Layer(ALP - i) = Layer(2) \cup Layer(1) \cup Layer(0) \\ &= \{FogLayer_2, FogLayer_1, EdgeLayer\} \\ View_2(3,3) :&= \bigcup_{i=0}^{n-1} Layer(ALP - i) = Layer(3) \cup Layer(2) \cup Layer(1) \\ &= \{CloudLayer, FogLayer_2, FogLayer1\} \end{aligned}$$

$View_1$ includes the *Fog Layer 2*, *Fog Layer 1*, and *Edge Layer*. Its focus is set to individual devices and how their data is used to support their movement throughout the environment. This View is of particular interest for the drones themselves which can avoid collisions, and for the city officials who get an overview of the drone movement in their city.

$View_2$ includes the *Cloud Layer*, *Fog Layer 2*, and *Fog Layer 1* while the *Edge Layer*, and thus the Edge Devices are abstracted away. Using this View, the aggregated movement and movement patterns can be described without focusing on individual devices.

The two Views in Figure 6.24 highlight one Fog Component (①), which changes its Fog Type based on the selected View. While the first View assumes the Fog Component which is a city access point to be a Fog Node, and therefore provide services to the connected drones, the second View abstracts away the individual drones and comprises them into the access points. Therefore, this View assumes that the access points behave as if they were on the *Edge Layer* only providing data.

DisCoFog 2: The second part of this case study are the results for *DisCoFog 2*. DisCoFog 2 was placed within the smart environment domain, but looked at the scalable aspect which was not covered by ARControl. The initial Fog Architecture setup of DisCoFog 2 is shown in Figure 6.25. It started from the same setup as used for ARControl, as both were placed within the Intelligent Workplace[31], except without the Occupant Smartphone which was dynamically added for the ARControl case study. For the sake of completeness and the independence of the ARControl case study, we briefly describe the setup as done for each case study.

[31]https://www.cmu.edu/homepage/innovation/2007/spring/intelligent-workplace.shtml

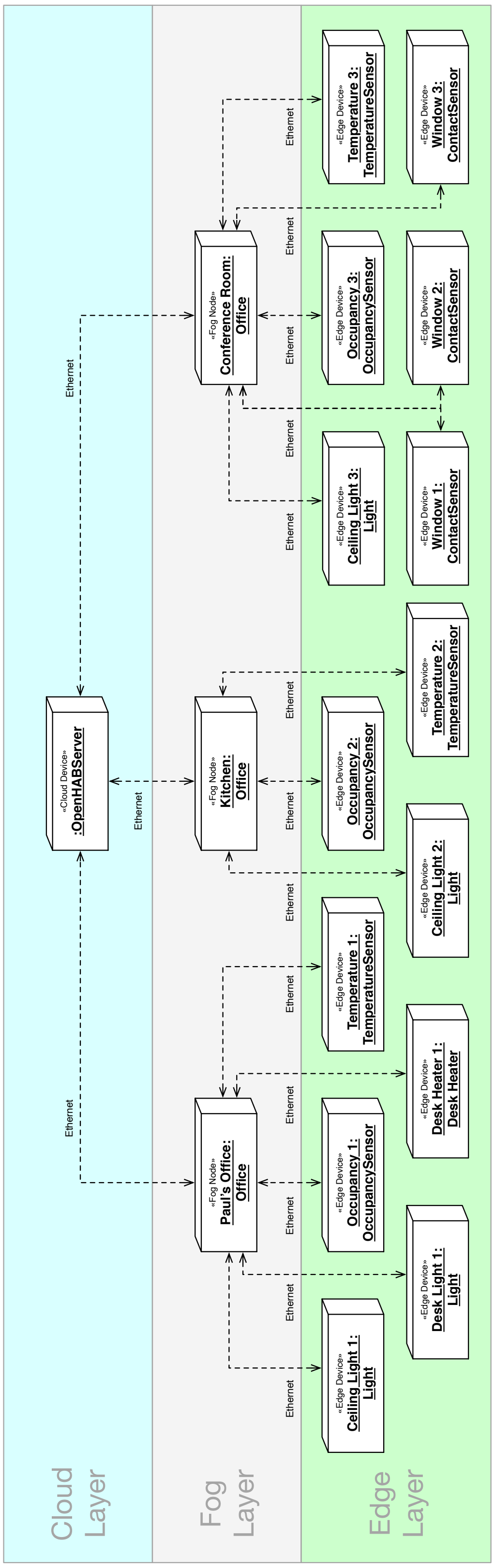

Cloud Layer
Fog Layer
Edge Layer
«Cloud Device» :OpenHABServer
«Fog Node» Paul's Office: Office
«Fog Node» Kitchen: Office
«Fog Node» Conference Room: Office
«Edge Device» Ceiling Light 1: Light
«Edge Device» Desk Light 1: Light
«Edge Device» Occupancy 1: OccupancySensor
«Edge Device» Desk Heater 1: Desk Heater
«Edge Device» Temperature 1: TemperatureSensor
«Edge Device» Ceiling Light 2: Light
«Edge Device» Occupancy 2: OccupancySensor
«Edge Device» Temperature 2: TemperatureSensor
«Edge Device» Ceiling Light 3: Light
«Edge Device» Window 1: ContactSensor
«Edge Device» Occupancy 3: OccupancySensor
«Edge Device» Window 2: ContactSensor
«Edge Device» Temperature 3: TemperatureSensor
«Edge Device» Window 3: ContactSensor
Ethernet

Figure 6.25: The HW/SW mapping of the DisCoFog 2 setup (UML Deployment Diagram). For readability, we only show components and the used communication channels. DisCoFog 2 is a 3-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers.

The Fog Architecture in Figure 6.25 contained three layers, three tiers, respectively. On the *Cloud Layer*, one single Fog Component was responsible for the coordination and control of the smart environment. This Fog Component was an instance of the OpenHAB[32] project, an open source project that tries to integrate all devices which are used within smart environments. The `OpenHABServer` was established as the central control unit in the Intelligent Workplace in an iPraktikum[33] project as part of a cooperation between Carnegie Mellon University and Technical University of Munich in the year 2014.

The *Fog Layer* contained three Fog Components that represented the connected offices: `Paul's Office`, `Kitchen`, and `Conference Room`. Each Fog Component was a Raspberry Pi[34] that allowed to establish a wired or wireless connection which was used to connect different types of sensors and actuators as well as interaction devices such as the `Occupant Smartphone` as shown in Section 6.2.1.

The *Edge Layer* included these sensors and actuators. Each room Fog Node shown on the *Fog Layer* contained different connected devices based on the rooms usage. In `Paul's Office` more personalized devices such as a `Desk Light 1` could be found than in the `Kitchen` or `Conference Room`. Those rooms, that were designed for a broad spectrum of occupants, were reduced to include more generic devices.

Equation 57 shows the definition for this Fog Architecture as a set of layers: The *Edge Layer*, *Fog Layer*, and *Cloud Layer*. As the Fog Layer Set only included a single Fog Layer, the cardinality of the Fog Architecture is equal to three. a

Equation 57: DisCoFog 2: Fog Architecture

$$\mathsf{FogArchitecture} := \{\mathsf{EdgeLayer},\ \mathsf{FogLayer},\ \mathsf{CloudLayer}\}$$
$$|\mathsf{FogArchitecture}| = 2 + |\mathsf{FogLayerSet}| = 3$$

The Service Set is shown in Equation 58 and contains six services. Three global services that were offered by the `OpenHABServer`, and thus the cloud which was reachable even from outside of the smart environment: `getGlobalDeviceList`, `getGlobalDeviceValue`, and `setGlobalDeviceValue`. The Fog Layer, on the other hand, included services that were available for more localized uses and only provided access to the devices within the connected room: `getDeviceList`, `getDeviceValue`, and `setDeviceValue`.

[32]https://www.openhab.org
[33]https://ase.in.tum.de/lehrstuhl_1/component/content/article/42-projects/current-projects/555-ios-praktikum-2014?Itemid=115
[34]https://www.raspberrypi.org

This Service Set Mapping is also shown in Equation 58 with the Cloud Layer not requesting any services, the Fog Layer being the connection between Cloud Layer and Edge Layer, providing and using the according services, and the Edge Layer only using local services provided by the Fog Layer. These relations are shown as a set of triples:

$$(\{ConsumedServices\}, Layer, \{ProvidedServices\})$$

Equation 58: DisCoFog 2: Service Set and Mapping

$$\begin{aligned}
ServiceSet = \{&getGlobalDeviceList, getGlobalDeviceValue,\\
&setGlobalDeviceValue, getDeviceList,\\
&getDeviceValue, setDeviceValue\}\\
ServiceSetMapping = \{&\\
&(\{\}, CloudLayer, \{getGlobalDeviceList,\\
&\quad getGlobalDeviceValue, setGlobalDeviceValue\}),\\
&(\{getGlobalDeviceList, getGlobalDeviceValue, setGlobalDeviceValue\},\\
&FogLayer, \{getDeviceList, getDeviceValue, setDeviceValue\}),\\
&(\{getDeviceList, getDeviceValue, setDeviceValue\}, EdgeLayer, \{\})\\
\}&
\end{aligned}$$

The Service Set Mapping is also represented in the Communication Set shown in Equation 59. The Cloud Layer was connected to the Raspberry Pis that were placed within each room by `Ethernet`. While the Raspberry Pis would have also allowed WIFI connections for sensors and actuators, they were all connected via ethernet which is why we reduced the Communication Set for readability to aggregate the connections to all sensors and actuators to a pseudo `Edge Device`. The entire Communication Set can be found in the appendix Section B.5 together with the Fog Set and Fog Component to Layer mapping.

Equation 59: DisCoFog 2: Communication Set

$$\begin{aligned}
CommunicationSet = \{&(OpenHABServer, Etherent, Paul'sOffice)^{\leftrightarrow},\\
&(OpenHABServer, Etherent, Kitchen)^{\leftrightarrow},\\
&(OpenHABServer, Etherent, ConferenceRoom)^{\leftrightarrow},\\
&(Paul'sOffice, Etherent, EdgeDevice)^{\leftrightarrow},\\
&(Kitchen, Etherent, EdgeDevice)^{\leftrightarrow},\\
&(ConferenceRoom, Etherent, EdgeDevice)^{\leftrightarrow}\}
\end{aligned}$$

After describing the initial Fog Architecture setup and the most important sets for scalable Fog Architectures for the DisCoFog 2 case study, Figure 6.26 shows the DisCoFog 2 setup after the addition of the new Fog Layer (①). This new layer (*Fog Layer 2*) was placed between the *Cloud Layer* and the former *Fog Layer* which is referred to as *Fog Layer 1* after the addition of the new Fog Layer. *Fog Layer 2* included one new Fog Component called `Floor 1` which aggregated the data of the connected rooms. Introducing this layer allowed an improved scalability of the previous setup to be extensible to other departments on other floors within the same building.

To incorporate these changes to the Fog Architecture, we have to adjust the Fog Architecture definition which in turn results in a change of its cardinality, the Service Set to mirror the newly offered services of *Fog Layer 2*, and the Service Set Mapping as shown in Section 4.2.2. Finally, the Communication Set changes based on the new connections between the rooms and the `Floor 1` Fog Node.

The changes are shown in Equation 60. For the Fog Architecture, we remove the previous *Fog Layer* which is not a unique description of the layer any more and add the new descriptor *Fog Layer 1* as well as the new *Fog Layer 2*.

The cardinality of the Fog Architecture changed to four with an additional layer within the Fog Layer Set. To mimic a new Cloud Layer from the perspective of *Fog Layer 1*, the new *Fog Layer 2* had to offer the services previously offered by the *Cloud Layer*. Thus, from the perspective of the service consumers of the previous *Cloud Layer*, nothing had changed except the communication partner. Meanwhile, the services offered by the *Cloud Layer* were adjusted to reflect these changes. This change was reflected by the three new services: $\mathtt{getGlobalDeviceList}^*$, $\mathtt{getGlobalDeviceValue}^*$, and $\mathtt{setGlobalDeviceList}^*$. These services were used by the new *Fog Layer 2* to propagate the information that they gathered from the rooms.

Accordingly, in the Service Set Mapping, we remove the mapping that relates to the *Cloud Layer* and the previous *Fog Layer*, update the new mappings for the *Cloud Layer* and Fog Layer 1, and add the mapping for *Fog Layer 2*. In the Communication Set we remove the Communication Components that describe the connections between the `OpenHABServer` instance and the room Fog Nodes as the `OpenHABServer` was not offering the services requested by those Fog Nodes any more. Additionally, we have to add the new Communication Components between the `OpenHABServer` and the rooms to the new Fog Component: `Floor`.

The changes to the Fog Set while not being the focus in this validation are in the appendix Section B.5 as well as the written out version of the Communication Set without the introduced abbreviations.

After the layer addition and the adjustments to the sets which describe the Fog Architecture, we had a 4-layered Fog Architecture which allows us to highlight two different Abstraction Levels for 3-layered Views.

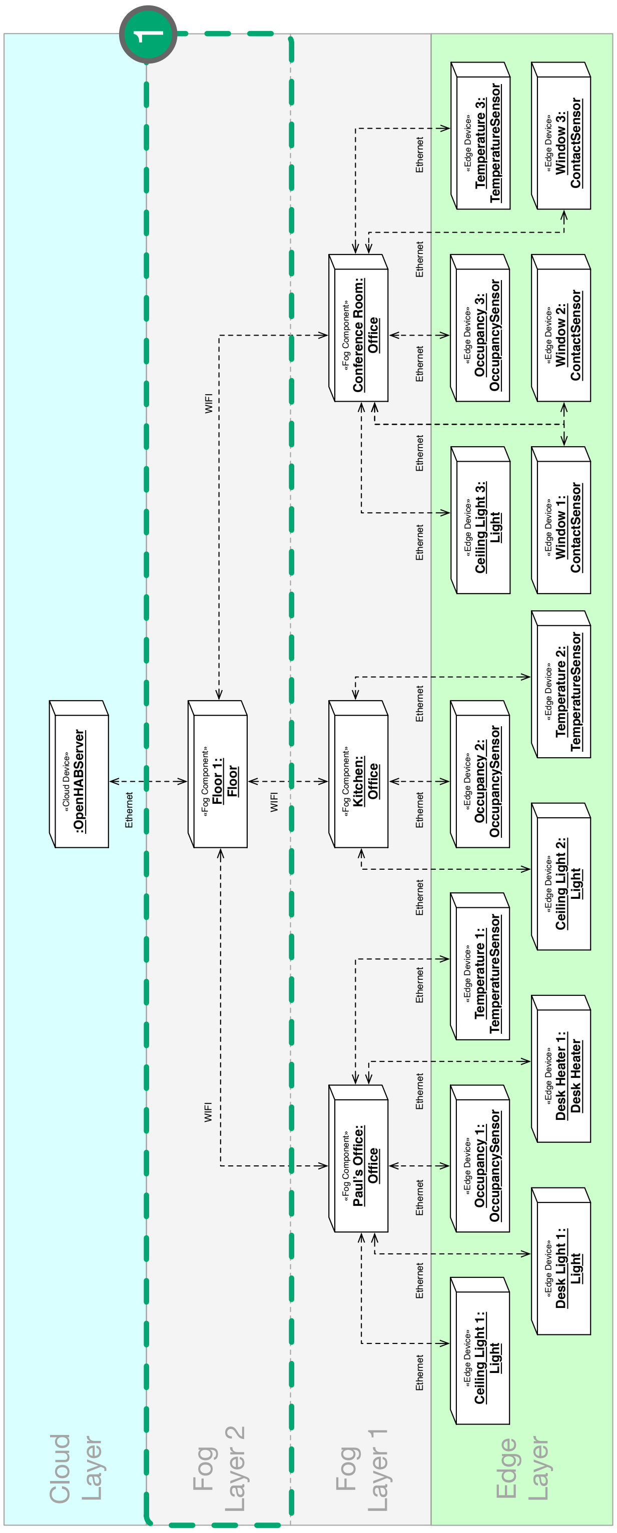

1
Cloud Layer
Fog Layer 2
Fog Layer 1
Edge Layer
«Cloud Device» :OpenHABServer
Ethernet
«Fog Component» Floor 1: Floor
WIFI
«Fog Component» Paul's Office: Office
«Fog Component» Kitchen: Office
«Fog Component» Conference Room: Office
«Edge Device» Ceiling Light 1: Light
«Edge Device» Desk Light 1: Light
«Edge Device» Occupancy 1: OccupancySensor
«Edge Device» Desk Heater 1: Desk Heater
«Edge Device» Temperature 1: TemperatureSensor
«Edge Device» Ceiling Light 2: Light
«Edge Device» Occupancy 2: OccupancySensor
«Edge Device» Temperature 2: TemperatureSensor
«Edge Device» Ceiling Light 3: Light
«Edge Device» Window 1: ContactSensor
«Edge Device» Occupancy 3: OccupancySensor
«Edge Device» Window 2: ContactSensor
«Edge Device» Temperature 3: TemperatureSensor
«Edge Device» Window 3: ContactSensor

Figure 6.26: The HW/SW mapping of the DisCoFog 2 setup after adding a new layer (UML Deployment Diagram). For readability, we only show components and the used communication channels. It shows a 4-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer 2*, *Fog Layer 1*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers. ① highlights the added layer and the `Floor 1` Fog Node.

These Views are shown in Figure 6.27 in which ① highlights a Fog Component which dynamically changes its type from a Cloud Device in the first View and a Fog Node in the second View.

Equation 60: DisCoFog 2: Layer Addition

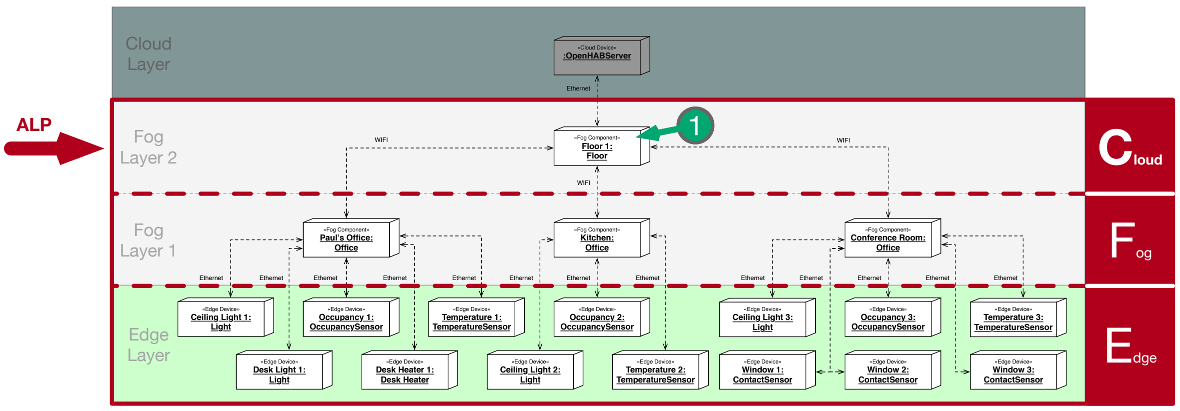
$$\begin{aligned}
\mathtt{FogArchitecture}_{new} &= \mathtt{FogArchitecture}_{old} \setminus \{\mathtt{FogLayer}\} \cup \\
&\quad \{\mathtt{FogLayer1}, \mathtt{FogLayer2}\}
\end{aligned}$$

$$|\mathtt{FogArchitecture}| = 2 + |\mathtt{FogLayerSet}| = 4$$

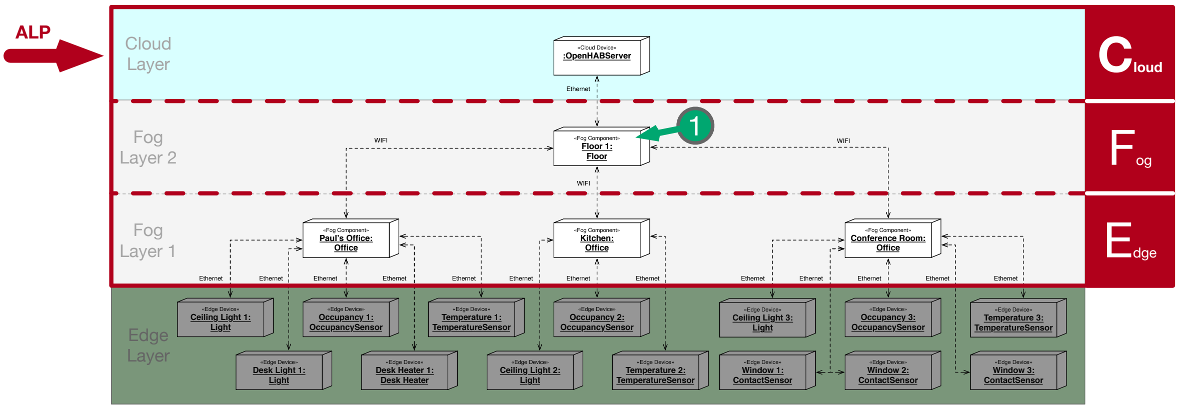
$$\begin{aligned}
\mathtt{ServiceSet}_{new} &= \mathtt{ServiceSet}_{old} \cup \{\mathtt{getGlobalDeviceList}^*, \\
&\quad \mathtt{getGlobalDeviceValue}^*, \mathtt{setGlobalDeviceValue}^*\}
\end{aligned}$$

$$\begin{aligned}
&\mathtt{ServiceSetMapping}_{new} = \\
&\quad \mathtt{ServiceSetMapping}_{old} \setminus \{(\{\}, \mathtt{CloudLayer}, \\
&\quad \{\mathtt{getGlobalDeviceList}, \mathtt{getGlobalDeviceValue}, \mathtt{setGlobalDeviceValue}\}), \\
&\quad (\{\mathtt{getGlobalDeviceList}, \mathtt{getGlobalDeviceValue}, \mathtt{setGlobalDeviceValue}\}, \\
&\quad \mathtt{FogLayer}, \{\mathtt{getDeviceList}, \mathtt{getDeviceValue}, \mathtt{setDeviceValue}\})\} \\
&\quad \cup \\
&\quad \{(\{\}, \mathtt{CloudLayer}, \{\mathtt{getGlobalDeviceList}^*, \mathtt{getGlobalDeviceValue}^*, \\
&\qquad \mathtt{setGlobalDeviceValue}^*\}), \\
&\quad (\{\mathtt{getGlobalDeviceList}^*, \mathtt{getGlobalDeviceValue}^*, \\
&\qquad \mathtt{setGlobalDeviceValue}^*\}, \mathtt{FogLayer2}, \\
&\quad \{\mathtt{getGlobalDeviceList}, \mathtt{getGlobalDeviceValue}, \mathtt{setGlobalDeviceValue}\}), \\
&\quad (\{\mathtt{getGlobalDeviceList}, \mathtt{getGlobalDeviceValue}, \mathtt{setGlobalDeviceValue}\}, \\
&\quad \mathtt{FogLayer1}, \{\mathtt{getDeviceList}, \mathtt{getDeviceValue}, \mathtt{setDeviceValue}\})\}
\end{aligned}$$

$$\begin{aligned}
\mathtt{CommunicationSet} = \mathtt{CommunicationSet}_{old} \setminus \\
&\{(\mathtt{OpenHABServer}, \mathtt{Ethernet}, \mathtt{Paul'sOffice})^{\leftrightarrow}, \\
&(\mathtt{OpenHABServer}, \mathtt{Ethernet}, \mathtt{Kitchen})^{\leftrightarrow}, \\
&(\mathtt{OpenHABServer}, \mathtt{Ethernet}, \mathtt{ConferenceRoom})^{\leftrightarrow}\} \\
&\cup \\
&\{(\mathtt{OpenHABServer}, \mathtt{Ethernet}, \mathtt{Floor1})^{\leftrightarrow}, \\
&(\mathtt{Floor1}, \mathtt{WIFI}, \mathtt{Paul'sOffice})^{\leftrightarrow}, \\
&(\mathtt{Floor1}, \mathtt{WIFI}, \mathtt{Kitchen})^{\leftrightarrow}, \\
&(\mathtt{Floor1}, \mathtt{WIFI}, \mathtt{ConferenceRoom})^{\leftrightarrow}\}
\end{aligned}$$


ALP
Cloud Layer
«Cloud Device» :OpenHABServer
Ethernet
Fog Layer 2
WIFI
«Fog Component» Floor 1: Floor
1
WIFI
WIFI
Fog Layer 1
«Fog Component» Paul's Office: Office
«Fog Component» Kitchen: Office
«Fog Component» Conference Room: Office
Ethernet
Edge Layer
«Edge Device» Ceiling Light 1: Light
«Edge Device» Occupancy 1: OccupancySensor
«Edge Device» Temperature 1: TemperatureSensor
«Edge Device» Occupancy 2: OccupancySensor
«Edge Device» Ceiling Light 3: Light
«Edge Device» Occupancy 3: OccupancySensor
«Edge Device» Temperature 3: TemperatureSensor
«Edge Device» Desk Light 1: Light
«Edge Device» Desk Heater 1: Desk Heater
«Edge Device» Ceiling Light 2: Light
«Edge Device» Temperature 2: TemperatureSensor
«Edge Device» Window 1: ContactSensor
«Edge Device» Window 2: ContactSensor
«Edge Device» Window 3: ContactSensor
Cloud
Fog
Edge

(a) View$_1$: The first 3-layered View of the Fog Architecture. The Abstraction Level is set to the third layer which relates to the `Fog Layer 2`.


ALP
Cloud Layer
«Cloud Device» :OpenHABServer
Ethernet
Fog Layer 2
WIFI
«Fog Component» Floor 1: Floor
1
WIFI
WIFI
Fog Layer 1
«Fog Component» Paul's Office: Office
«Fog Component» Kitchen: Office
«Fog Component» Conference Room: Office
Ethernet
Edge Layer
«Edge Device» Ceiling Light 1: Light
«Edge Device» Occupancy 1: OccupancySensor
«Edge Device» Temperature 1: TemperatureSensor
«Edge Device» Occupancy 2: OccupancySensor
«Edge Device» Ceiling Light 3: Light
«Edge Device» Occupancy 3: OccupancySensor
«Edge Device» Temperature 3: TemperatureSensor
«Edge Device» Desk Light 1: Light
«Edge Device» Desk Heater 1: Desk Heater
«Edge Device» Ceiling Light 2: Light
«Edge Device» Temperature 2: TemperatureSensor
«Edge Device» Window 1: ContactSensor
«Edge Device» Window 2: ContactSensor
«Edge Device» Window 3: ContactSensor
Cloud
Fog
Edge

(b) View$_2$: The second 3-layered View of the Fog Architecture. The Abstraction Level is set to the fourth layer which relates to the `Cloud Layer`.

Figure 6.27: The HW/SW mapping of the DisCoFog 2 showing the different 3-layered Views (UML Deployment Diagram). For readability, we only show components and the used communication channels. It shows a 4-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer 2*, *Fog Layer 1*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers. ① highlights one example Fog Component which dynamically changes its *Fog Type* based on the View it is part of.

The definition of the Views is shown in Equation 61.

Equation 61: DisCoFog 2: Views

$$View(ALP, n) := \bigcup_{i=0}^{n-1} Layer(ALP - i)$$

$$\begin{aligned} View_1(2,3) :&= \bigcup_{i=0}^{n-1} Layer(ALP - i) = Layer(2) \cup Layer(1) \cup Layer(0) \\ &= \{FogLayer_2, FogLayer_1, EdgeLayer\} \\ View_2(3,3) :&= \bigcup_{i=0}^{n-1} Layer(ALP - i) = Layer(2) \cup Layer(1) \cup Layer(0) \\ &= \{CloudLayer, FogLayer_2, FogLayer1\} \end{aligned}$$

The $View_1$ is established by setting the Abstraction Level to Fog Layer 2, and therefore contains *Fog Layer 2*, *Fog Layer 1*, and the *Edge Layer*. This View shows the individual devices the occupant can interact with using their locally closest Fog Nodes. Additionally, with representing the `Floor 1` on the Cloud Layer, it highlights device usage within one floor, e. g., one department of the Intelligent Workplace.

In contrast, $View_2$ emphasizes the interconnectivity between the departments and allows comparisons between them, e. g., referring to the energy consumption and the usage of specific device types without going in too much detail about singular sensors or actuators. The second View includes the *Cloud Layer*, *Fog Layer 2*, and *Fog Layer 1*.

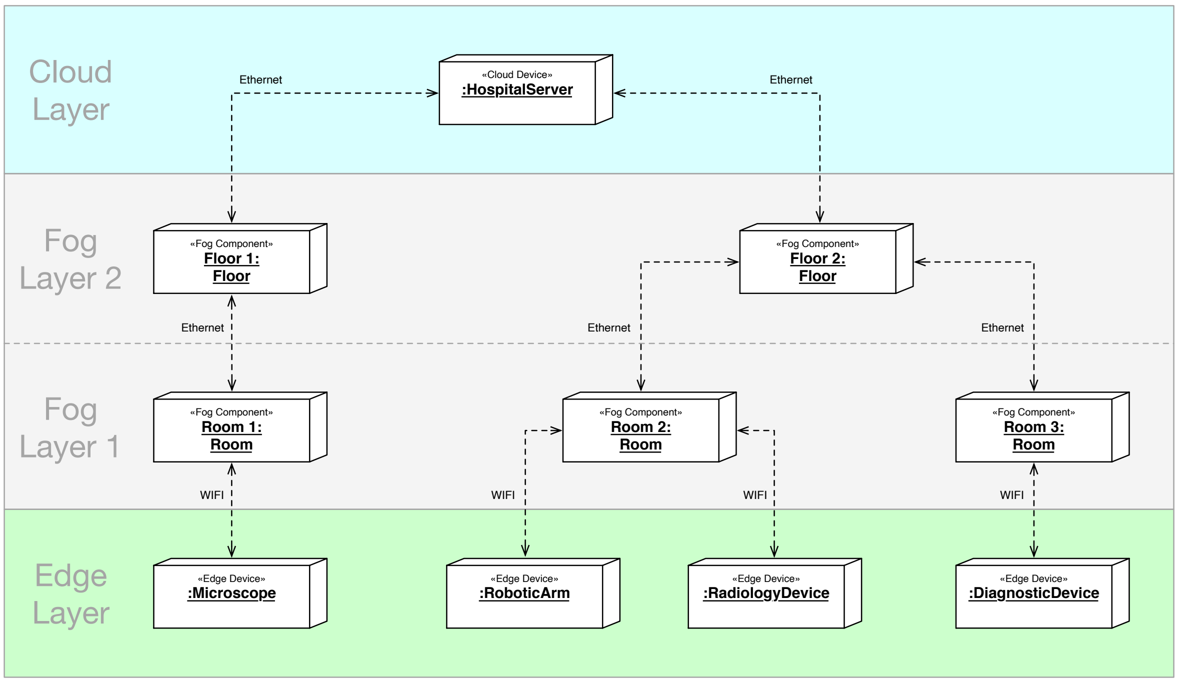

Cloud Layer
Fog Layer 2
Fog Layer 1
Edge Layer
«Cloud Device» :HospitalServer
Ethernet
Ethernet
«Fog Component» Floor 1: Floor
«Fog Component» Floor 2: Floor
Ethernet
Ethernet
Ethernet
«Fog Component» Room 1: Room
«Fog Component» Room 2: Room
«Fog Component» Room 3: Room
WIFI
WIFI
WIFI
WIFI
«Edge Device» :Microscope
«Edge Device» :RoboticArm
«Edge Device» :RadiologyDevice
«Edge Device» :DiagnosticDevice

Figure 6.28: The HW/SW mapping of the eHealth setup (UML Deployment Diagram). For readability, we only show components and the used communication channels. It shows a 4-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer 2*, *Fog Layer 1*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers.

eHealth

The second validation of the scalable Fog Architecture validation is *eHealth*. eHealth is placed within the health domain. The goal of the case study was to include the scalable Fog Architecture concept in a hospital setting. As before, we focus on the most important concepts and sets, but for completeness provide the Fog Set and its layers, the unabbreviated Communication Set, and the changes to the sets after the layer addition in the appendix in Section B.6. The Fog Architecture setup for eHealth is shown in Figure 6.28. It was a 4-layered Fog Architecture with two Fog Layers which were all deployed on different hardware components, and thus related to tiers.

The *Cloud Layer* contained the central `Hospital Server`. The `Hospital Server` was used to persist data about patients that was, e. g., used across several floors by different departments.

Each department, represented by a floor node on *Fog Layer 2*, used its own local software which was adjusted to the department's use cases and contained more specific data than the central instance. These floor nodes are typically servers by themselves due to computational power needed to run the medical software.

Fog Layer 1 contained three Fog Nodes: `Room 1`, `Room 2`, and `Room 3`. Each room node described one treatment room containing specialized medical equipment. They were used to preprocess and encrypt the gathered data and made it accessible for doctors who dynamically entered and left the Fog Architecture with their mobile devices. As the dynamic addition of Fog Components was not the focus of this case study, we are not going in too much detail about that process. Due to hospital restrictions, the connections between the Fog Components down to this level were all wired as indicated by `Ethernet`.

Finally, the *Edge Layer* contained the medical equipment needed in the treatment room. This equipment could either be connected wired or wireless depending on the equipment's specifications. The equipment could range from `Microscopes`, `Radiology Devices`, and `Diagnostic Devices`, to `Robotic Arms` and more.

The resulting Fog Architecture definition based on the included layers is shown in Equation 62. With two layers in the Fog Layer Set, the cardinality of the Fog Architecture was equal to four, according to the number of layers.

Equation 62: eHealth: Fog Architecture

$$\mathsf{FogArchitecture} := \{\mathsf{EdgeLayer},\ \mathsf{FogLayer1},\ \mathsf{FogLayer2},\ \mathsf{CloudLayer}\}$$
$$|\mathsf{FogArchitecture}| = 2 + |\mathsf{FogLayerSet}| = 4$$

Equation 63 shows the Service Set and the respective Service Set Mapping.

Equation 63: eHealth: Service Set and Mapping

$$\begin{aligned}\mathsf{ServiceSet} = \{&\mathsf{getGlobalPatientList}, \mathsf{getGlobalPatientData},\\ &\mathsf{setGlobalPatientData}, \mathsf{getDepartmentPatientList},\\ &\mathsf{getDepartmentPatientData}, \mathsf{setDepartmentPatientData},\\ &\mathsf{getRoomData}, \mathsf{setRoomData}\}\end{aligned}$$

$$\begin{aligned}&\mathsf{ServiceSetMapping} = \{\\ &\quad(\{\}, \mathsf{CloudLayer},\\ &\quad\{\mathsf{getGlobalPatientList}, \mathsf{getGlobalPatientData}, \mathsf{setGlobalPatientData}\}),\\ &\quad(\{\mathsf{getGlobalPatientList}, \mathsf{getGlobalPatientData}, \mathsf{setGlobalPatientData}\},\\ &\quad\mathsf{FogLayer2}, \{\mathsf{getDepartmentPatientList}, \mathsf{getDepartmentPatientData},\\ &\qquad\mathsf{setDepartmentPatientData}\}),\\ &\quad(\{\mathsf{getDepartmentPatientList}, \mathsf{getDepartmentPatientData},\\ &\qquad\mathsf{setDepartmentPatientData}\}, \mathsf{FogLayer1},\\ &\quad\{\mathsf{getRoomData}, \mathsf{setRoomData}\}),\\ &\quad(\{\mathsf{getRoomData}, \mathsf{setRoomData}\}, \mathsf{EdgeLayer}, \{\})\}\end{aligned}$$

The Service Set consists of eight services. Three of them were offered by the Hospital Server: getGlobalPatientList, getGlobalPatientData, and setGlobalPatientData. These services were used to access and manipulate the information the entire hospital had about a patient. In contrast, the services getDepartmentPatientList, getDepartmentPatientValue, and setDepartmentPatientValue accessed and changed the data which was stored in the system of the local department. The information stored and received from this system did not have to match the information provided by the Hospital Server. The last two services getRoomData and setRoomData provided access to the local data currently aggregated and used in a treatment room.

Accordingly, the Service Set Mapping shows the offered services by the *Cloud Layer*, the offered and consumed services of *Fog Layer 2* and *Fog Layer 1*, and the consumed services of the *Edge Layer*. The strict hierarchical mapping is also reflected into the Communication Set which is shown in Equation 64.

Equation 64: eHealth: Communication Set

$$\begin{aligned}
CommunicationSet = \{&(HospitalServer, Ethernet, Floor1)^{\leftrightarrow},\\
&(HospitalServer, Ethernet, Floor2)^{\leftrightarrow},\\
&(Floor1, Ethernet, Room1)^{\leftrightarrow},\\
&(Floor2, Ethernet, Room2)^{\leftrightarrow},\\
&(Floor2, Ethernet, Room3)^{\leftrightarrow},\\
&(Room1, WIFI, Microscope)^{\leftrightarrow},\\
&(Room2, WIFI, RoboticArm)^{\leftrightarrow},\\
&(Room2, WIFI, RadiologyDevice)^{\leftrightarrow},\\
&(Room3, WIFI, DiagnosticDevice)^{\leftrightarrow}\}
\end{aligned}$$

The Hospital Server was connected to both floor nodes via ethernet. Floor 1 was connected by ethernet to the two room nodes: Room 1 and Room Node 2, Floor 2 was connected to Room 3. Each room had individual equipment which was connected via WIFI. According to constraints set by the hospital, only the room nodes provided WIFI access.

While the previous sets describe the initial eHealth setup, Figure 6.29 shows the setup which should be established after adding a layer.

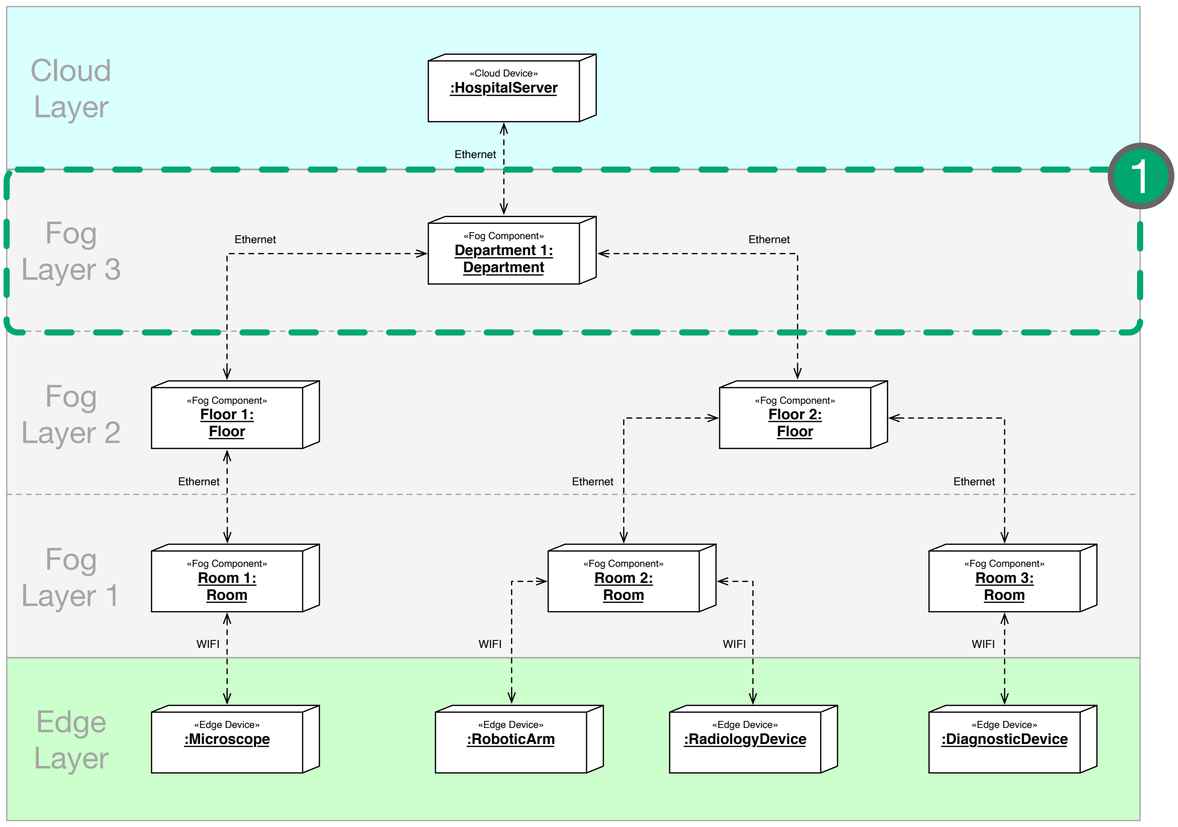

Cloud Layer
«Cloud Device» :HospitalServer
Ethernet
1
Fog Layer 3
«Fog Component» Department 1: Department
Ethernet
Ethernet
Fog Layer 2
«Fog Component» Floor 1: Floor
«Fog Component» Floor 2: Floor
Ethernet
Ethernet
Ethernet
Fog Layer 1
«Fog Component» Room 1: Room
«Fog Component» Room 2: Room
«Fog Component» Room 3: Room
WIFI
WIFI
WIFI
WIFI
Edge Layer
«Edge Device» :Microscope
«Edge Device» :RoboticArm
«Edge Device» :RadiologyDevice
«Edge Device» :DiagnosticDevice

Figure 6.29: The HW/SW mapping of the eHealth setup after adding a new layer (UML Deployment Diagram). For readability, we only show components and the used communication channels. It shows a 5-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer 3*, *Fog Layer 2*, *Fog Layer 1*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers. ① highlights the added layer and the contained `Department 1`.

As highlighted by ①, this setup contains an additional layer with a Fog Component called `Department 1`. The new layer was placed between the *Cloud Layer* and the topmost Fog Layer (*Fog Layer 2*). In the previous setup, the floor nodes described the department level. We wanted to add the `Department 1` as some departments in the hospital were spread across multiple floors which represented different parts of the department with different required information which was requested from the same information system.

The needed adjustments to the existing setup are shown in Equation 65. The Fog Architecture was extended by one additional layer: `Fog Layer 3`, which raised its cardinality to five. The new services that described the new layer are shown in the updated Service Set. In contrast to the DisCoFog use case, we do not take over the services offered by the layer which is on top of the newly added layer, but provide new services which are based on the services provided by the layer below.

As the required functionality, and thus service description is closely linked to the services already provided by the layer below, this layer which aggregates information should reflect this. For the Service Set Mapping, we have to remove the mapping for *Fog Layer 2* which no longer accesses the services provided by the *Cloud Layer*. These services are used by the new *Fog Layer 3* which in turn offers a new service to *Fog Layer 2*. Finally, the Communication Set needs to be adjusted. We remove the Communication Components that reflect the connection between *Cloud Layer* and *Fog Layer 2* but introduce a Communication Component which uses Ethernet to connect the `Hospital Server` with `Department 1` and two Communication Components that connect `Department 1` to the floor nodes.

Equation 65: eHealth: Layer Addition

$$\begin{aligned}
&\mathsf{FogArchitecture}_{new} = \mathsf{FogArchitecture}_{old} \cup \{\mathsf{FogLayer3}\}\\
&|\mathsf{FogArchitecture}| = 2 + |\mathsf{FogLayerSet}| = 5\\
&\mathsf{ServiceSet}_{new} = \mathsf{ServiceSet}_{old} \cup \{\mathsf{getDepartmentPatientList}^*,\\
&\qquad\qquad\qquad \mathsf{getDepartmentPatientData}^*,\\
&\qquad\qquad\qquad \mathsf{setDepartmentPatientData}^*\}\\
&\mathsf{ServiceSetMapping}_{new} = \mathsf{ServiceSetMapping}_{old} \setminus\\
&\quad (\{\mathsf{getGlobalPatientList}, \mathsf{getGlobalPatientData}, \mathsf{setGlobalPatientData}\},\\
&\quad \mathsf{FogLayer2}, \{\mathsf{getDepartmentPatientList},\\
&\qquad \mathsf{getDepartmentPatientData}, \mathsf{setDepartmentPatientData}\}) \cup\\
&\quad \{(\{\mathsf{getGlobalPatientList}, \mathsf{getGlobalPatientData},\\
&\qquad \mathsf{setGlobalPatientData}\}, \mathsf{FogLayer3}, \{\mathsf{getDepartmentPatientList}^*,\\
&\qquad \mathsf{getDepartmentPatientData}^*, \mathsf{setDepartmentPatientData}^*\}),\\
&\quad (\{\mathsf{getDepartmentPatientList}^*, \mathsf{getDepartmentPatientData}^*,\\
&\qquad \mathsf{setDepartmentPatientData}^*\}, \mathsf{FogLayer2},\\
&\quad \{\mathsf{getDepartmentPatientList}, \mathsf{getDepartmentPatientData},\\
&\qquad \mathsf{setDepartmentPatientData}\})\}\\
&\mathsf{CommunicationSet} = \mathsf{CommunicationSet}_{old} \setminus\\
&\qquad\qquad \{(\mathsf{HospitalServer}, \mathsf{Ethernet}, \mathsf{Floor1})^{\leftrightarrow},\\
&\qquad\qquad (\mathsf{HospitalServer}, \mathsf{Ethernet}, \mathsf{Floor2})^{\leftrightarrow}\}\\
&\qquad\qquad \cup\\
&\qquad\qquad \{(\mathsf{HospitalServer}, \mathsf{Ethernet}, \mathsf{Department1})^{\leftrightarrow},\\
&\qquad\qquad (\mathsf{Department1}, \mathsf{Ethernet}, \mathsf{Floor1})^{\leftrightarrow},\\
&\qquad\qquad (\mathsf{Department1}, \mathsf{Ethernet}, \mathsf{Floor2})^{\leftrightarrow}\}
\end{aligned}$$

Based on this additional layer, Figure 6.30 shows the three 3-layered Views which are contained within the new Fog Architecture. Each example contains the layers: *Cloud Layer*, *Fog Layer 3*, *Fog Layer 2*, *Fog Layer 1*, and the *Edge Layer*. Additionally, one Fog Component that dynamically changes its Fog Type based on the View layer assignment is shown with ①. Each View describes a different aspect of the Fog Architecture set by the Abstraction Level which can be further investigated.

The definitions of each of the three Views is given in Equation 66. $View_1$ contains the *Edge Layer*, and therefore describes the lowest possible 3-layered View. Its point of interest is the individual medical equipment which is spread across the different treatment rooms. It describes how the equipment can store information and how this information can be accessed by doctors. Additionally, it describes how this information is persisted in the information system provided on each floor node.

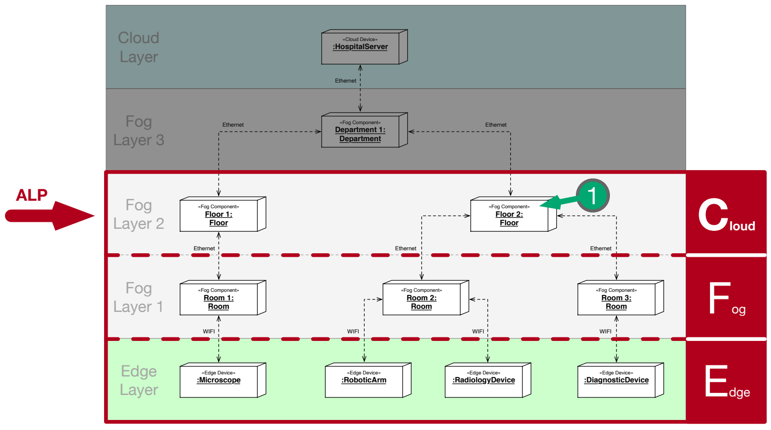
$View_2$ focuses on the information flow between the treatment rooms up to the department level. It describes which information is persisted in the local floor node, and thus accessible for the current part of the department and which information needs to be published within the entire department.

Finally, $View_3$ highlights the transition from the department level to the *Cloud Layer*. It describes which information is persisted in what way to be accessible by the entire hospital information system. Depending on the compliance of the information systems, the information might change quite significantly.

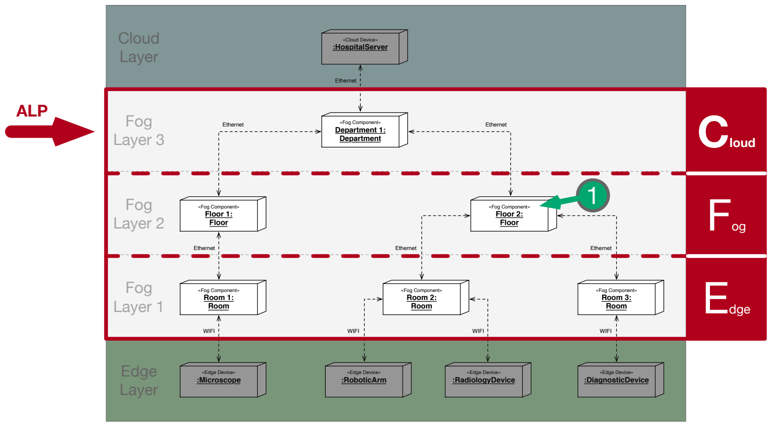
Equation 66: eHealth: Views

$$View(ALP, n) := \bigcup_{i=0}^{n-1} Layer(ALP - i)$$

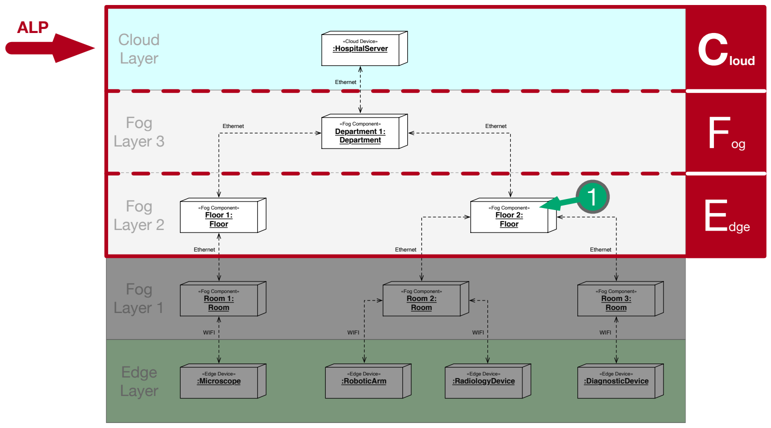
$$\begin{aligned}
View_1(2,3) &:= \bigcup_{i=0}^{n-1} Layer(ALP - i) = Layer(2) \cup Layer(1) \cup Layer(0) \\
&= \{FogLayer_2, FogLayer_1, EdgeLayer\} \\
View_2(3,3) &:= \bigcup_{i=0}^{n-1} Layer(ALP - i) = Layer(3) \cup Layer(2) \cup Layer(1) \\
&= \{FogLayer_3, FogLayer_2, FogLayer1\} \\
View_3(4,3) &:= \bigcup_{i=0}^{n-1} Layer(ALP - i) = Layer(4) \cup Layer(3) \cup Layer(2) \\
&= \{CloudLayer, FogLayer_3, FogLayer2\}
\end{aligned}$$


Cloud Layer
«Cloud Device» :HospitalServer
Ethernet
Fog Layer 3
Ethernet
«Fog Component» Department 1: Department
Ethernet
ALP
Fog Layer 2
«Fog Component» Floor 1: Floor
«Fog Component» Floor 2: Floor
1
Cloud
Ethernet
Ethernet
Ethernet
Fog Layer 1
«Fog Component» Room 1: Room
«Fog Component» Room 2: Room
«Fog Component» Room 3: Room
Fog
WIFI
WIFI
WIFI
WIFI
Edge Layer
«Edge Device» :Microscope
«Edge Device» :RoboticArm
«Edge Device» :RadiologyDevice
«Edge Device» :DiagnosticDevice
Edge

(a) View$_1$: The first 3-layered View of the Fog Architecture. The Abstraction Level is set to the third layer which relates to the `Fog Layer 2`.


Cloud Layer
«Cloud Device» :HospitalServer
Ethernet
ALP
Fog Layer 3
Ethernet
«Fog Component» Department 1: Department
Ethernet
Cloud
Fog Layer 2
«Fog Component» Floor 1: Floor
«Fog Component» Floor 2: Floor
1
Fog
Ethernet
Ethernet
Ethernet
Fog Layer 1
«Fog Component» Room 1: Room
«Fog Component» Room 2: Room
«Fog Component» Room 3: Room
Edge
WIFI
WIFI
WIFI
WIFI
Edge Layer
«Edge Device» :Microscope
«Edge Device» :RoboticArm
«Edge Device» :RadiologyDevice
«Edge Device» :DiagnosticDevice

(b) View$_2$: The second 3-layered View of the Fog Architecture. The Abstraction Level is set to the fourth layer which relates to the `Fog Layer 3`.


ALP
Cloud Layer
«Cloud Device» :HospitalServer
Cloud
Ethernet
Fog Layer 3
Ethernet
«Fog Component» Department 1: Department
Ethernet
Fog
Fog Layer 2
«Fog Component» Floor 1: Floor
«Fog Component» Floor 2: Floor
1
Edge
Ethernet
Ethernet
Ethernet
Fog Layer 1
«Fog Component» Room 1: Room
«Fog Component» Room 2: Room
«Fog Component» Room 3: Room
WIFI
WIFI
WIFI
WIFI
Edge Layer
«Edge Device» :Microscope
«Edge Device» :RoboticArm
«Edge Device» :RadiologyDevice
«Edge Device» :DiagnosticDevice

(c) View$_3$: The third 3-layered View of the Fog Architecture. The Abstraction Level is set to the fifth layer which relates to the `Cloud Layer`.

Figure 6.30: The HW/SW mapping of the eHealth showing the different 3-layered Views (UML Deployment Diagram). For readability, we only show components and the used communication channels. It shows a 5-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer 3*, *Fog Layer 2*, *Fog Layer 1*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers. ① highlights one example Fog Component which dynamically changes its *Fog Type* based on the View it is part of.

⑥ highlights the same Fog Component in every View. Based on the 3-layered View concept which describes a Fog Architecture on its own, the Fog Component dynamically changes its Fog Type based on the current View. In the Fog Architecture, the floor nodes are particularly interesting for the dynamic change as they change from a Cloud Device in $View_1$ to a Fog Node in $View_2$ and finally to an Edge Device in $View_3$.

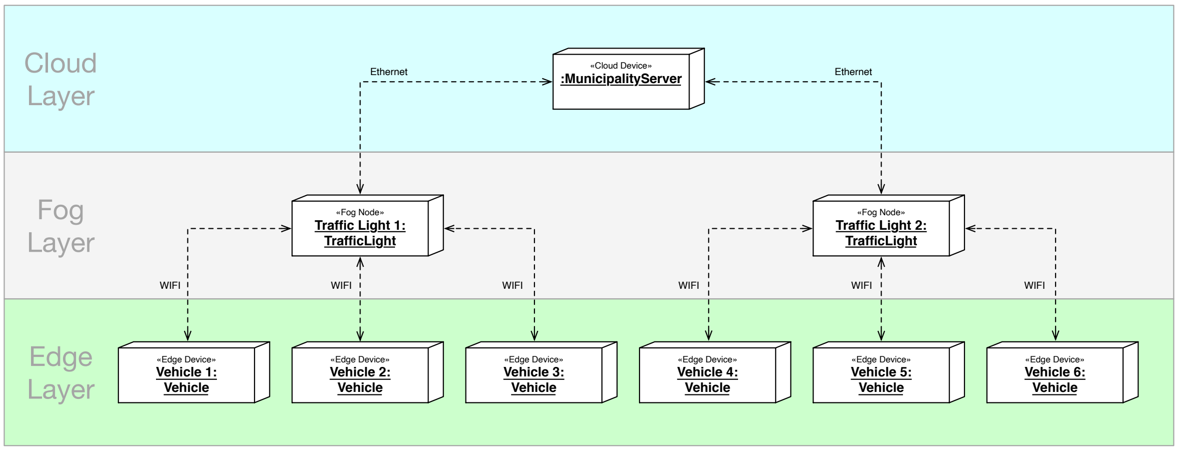

Cloud Layer
Fog Layer
Edge Layer
Ethernet
Ethernet
«Cloud Device» :MunicipalityServer
«Fog Node» Traffic Light 1: TrafficLight
«Fog Node» Traffic Light 2: TrafficLight
WIFI
«Edge Device» Vehicle 1: Vehicle
«Edge Device» Vehicle 2: Vehicle
«Edge Device» Vehicle 3: Vehicle
«Edge Device» Vehicle 4: Vehicle
«Edge Device» Vehicle 5: Vehicle
«Edge Device» Vehicle 6: Vehicle

Figure 6.31: The HW/SW mapping of the Fog.BOI setup (UML Deployment Diagram). For readability, we only show components and the used communication channels. It shows a 3-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers.

Fog.BOI

The last case study for the scalable Fog Architecture validation is *Fog.BOI* as introduced in Section 6.1.6. Similar to the DisCoFog 1 case study, it was also placed within the smart city context. The HW/SW mapping of Fog.BOI is shown in Figure 6.31. It describes a 3-layered Fog Architecture which emphasizes the locality to the vehicles in road traffic. The Fog Set and the layer definitions as well as the written out Communication Set before and after the addition of a new layer is shown in appendix Section B.7.

In the initial setup, the *Cloud Layer* contained one Fog Component, the `Municipality Server` which gathers the positions of different vehicles to create a traffic heat map to find traffic congestions and propagate those to the connected vehicles.

The *Fog Layer* contained two Fog Components: `Traffic Light 1` and `Traffic Light 2`. These Fog Components are small computational units that opened up a local WIFI which other Fog Components could use to get traffic updates as well as information about other connected devices and decide which vehicle went through the crossing first.

The *Edge Layer* contained the vehicles that wanted to use the traffic updates provided by the traffic lights. These vehicles could be any traffic participant that could connect to the traffic lights, and thus provided their own position information to other traffic participants.

Equation 67 shows the according Fog Architecture definition with three layers which is its cardinality.

Equation 67: Fog.BOI: Fog Architecture

$$\mathsf{FogArchitecture} := \{\mathsf{EdgeLayer},\ \mathsf{FogLayer},\ \mathsf{CloudLayer}\}$$
$$|\mathsf{FogArchitecture}| = 2 + |\mathsf{FogLayerSet}| = 3$$

Based on the position updates, many additional services are imaginable. As we want to describe the scalable concepts, we focus on three services as shown in Equation 68: `getVehiclePositions`, `setPosition`, and `uploadVehiclePositions`.

`getVehiclePositions` allowed traffic participants to continuously request the positions of every other traffic participant in the local surrounding of the traffic light they were currently connected to. While this service was beneficial to human drivers by noticing them when another vehicle was approaching the same crossing, it was essential for autonomous cars to negotiate which vehicle passes the crossing first.

The second service `setPosition` was the related counterpart to the first service which allowed vehicles to report their current position to the local instance in exchange for the positions of the other vehicles.

Finally, the `uploadVehiclePositions` service allowed the municipality to create heat maps of all streets covered by traffic lights, and therefore Fog Nodes.

Equation 68: Fog.BOI: Service Set and Mapping

$$\begin{aligned}
&\mathsf{ServiceSet} = \{\mathsf{getVehiclePositions}, \mathsf{setPosition}, \mathsf{uploadVehiclePositions}\}\\
&\mathsf{ServiceSetMapping} = \{\\
&\quad(\{\}, \mathsf{CloudLayer}, \{\mathsf{uploadVehiclePositions}\}),\\
&\quad(\{\mathsf{uploadVehiclePositions}\}, \mathsf{FogLayer},\\
&\quad\{\mathsf{getVehiclePositions}, \mathsf{setPosition}\}),\\
&\quad(\{\mathsf{getVehiclePositions}, \mathsf{setPosition}\}, \mathsf{EdgeLayer}, \{\})\\
&\}
\end{aligned}$$

Accordingly, the Service Set Mapping shows that the Cloud Layer offered the `uploadVehiclePositions` service to generate the computational intense heat maps. This service was used by the *Fog Layer* which in turn offered the other two services to the Fog Components on the *Edge Layer*.

The Communication Set, shown in Equation 69, highlights the Communication Components that create the connections between the layers. The complete Communication Set is shown in appendix Section B.7. To create a network of nearby vehicles, the traffic lights established a WIFI hotspot which could be accessed by the vehicles. The traffic lights themselves were hard wired to the municipality.

Equation 69: Fog.BOI: Communication Set

$$\begin{aligned}\mathsf{CommunicationSet} = \{&(\mathsf{MunicipalityServer}, \mathsf{Ethernet}, \mathsf{TrafficLight1})^{\leftrightarrow},\\ &(\mathsf{MunicipalityServer}, \mathsf{Ethernet}, \mathsf{TrafficLight2})^{\leftrightarrow},\\ &(\mathsf{TrafficLight1}, \mathsf{WIFI}, \mathsf{Vehicle1})^{\leftrightarrow},\\ &(\mathsf{TrafficLight1}, \mathsf{WIFI}, \mathsf{Vehicle2})^{\leftrightarrow},\\ &(\mathsf{TrafficLight1}, \mathsf{WIFI}, \mathsf{Vehicle3})^{\leftrightarrow},\\ &(\mathsf{TrafficLight2}, \mathsf{WIFI}, \mathsf{Vehicle4})^{\leftrightarrow},\\ &(\mathsf{TrafficLight2}, \mathsf{WIFI}, \mathsf{Vehicle5})^{\leftrightarrow},\\ &(\mathsf{TrafficLight2}, \mathsf{WIFI}, \mathsf{Vehicle6})^{\leftrightarrow}\}\end{aligned}$$

Figure 6.32 shows an additional layer between the *Cloud Layer* and the *Fog Layer*. This additional layer (①) was used to structure bigger municipalities into smaller districts which could already locally analyze the traffic situation in their commuting area. Accordingly, the added Fog Component was called `City District`.

Equation 70 shows the changes that need to be made to the sets after the new layer was added to the Fog Architecture. As the new layer is above the existing *Fog Layer*, the new Fog Layer is called *Fog Layer 2* and added to the Fog Architecture. This layer should already locally analyze the traffic that was previously only analyzed on the Cloud Layer. Thus, a new service is added to the Service Set: $\mathsf{uploadVehiclePositions}^*$. As shown by the updated Service Set Mapping, this new service is now offered by the *Cloud Layer* which no longer offers `uploadVehiclePositions`. This service is moved to *Fog Layer 2*. Finally, the previously called *Fog Layer* is renamed to *Fog Layer 1* to create a unique naming. In the Communication Set, every Communication Component is removed which described the connection between *Fog Layer 1* and the *Cloud Layer*. Those connections are replaced by the connections between the *Cloud Layer* and *Fog Layer 2*, and *Fog Layer 2* and *Fog Layer 1* which both use Ethernet.

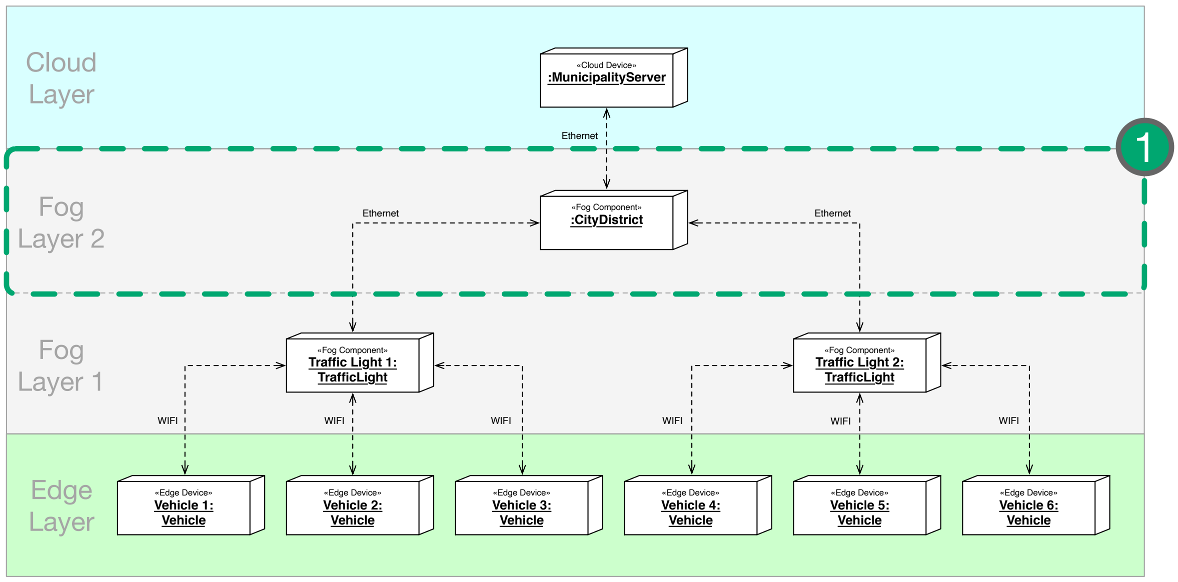
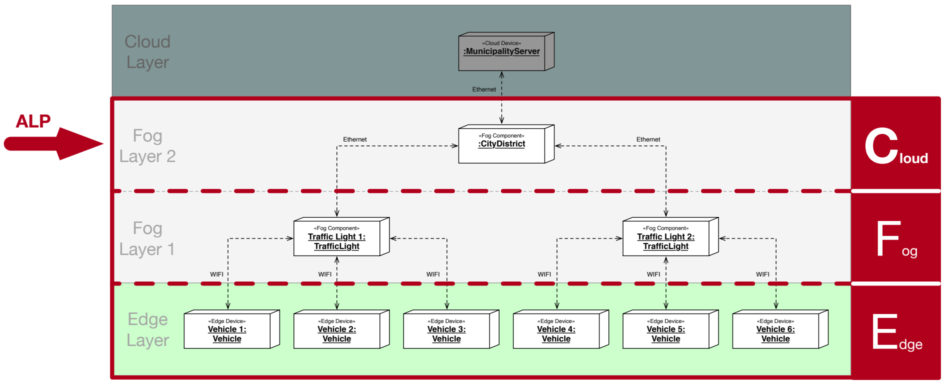


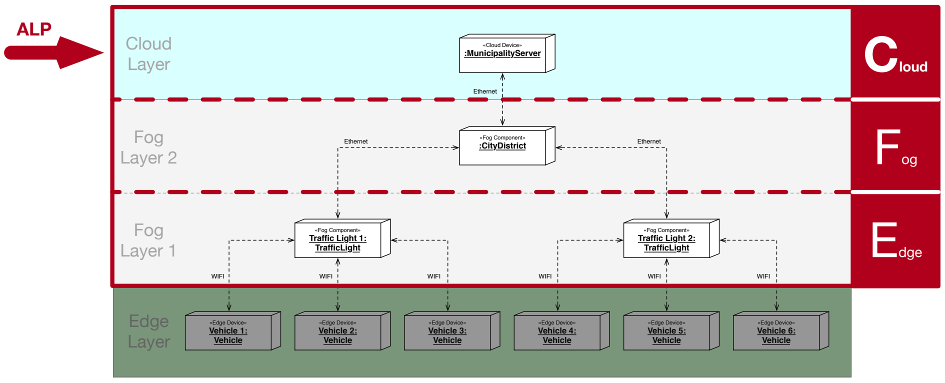
Figure 6.32: The HW/SW mapping of the Fog.BOI setup after adding a new layer (UML Deployment Diagram). For readability, we only show components and the used communication channels. It shows a 4-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer 2*, *Fog Layer 1*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers.

Equation 70: Fog.BOI: Layer Addition

$$\begin{aligned}
&\mathrm{FogArchitecture}_{new} = \mathrm{FogArchitecture}_{old} \cup \{\mathrm{FogLayer2}\}\\
&|\mathrm{FogArchitecture}| = 2 + |\mathrm{FogLayerSet}| = 4\\
&\mathrm{ServiceSet}_{new} = \mathrm{ServiceSet}_{old} \cup \{\mathrm{uploadVehiclePositions}^*\}\\
&\mathrm{ServiceSetMapping}_{new} = \mathrm{ServiceSetMapping}_{old} \setminus\\
&\quad \{(\{\}, \mathrm{CloudLayer}, \{\mathrm{uploadVehiclePositions}\}),\\
&\quad (\{\mathrm{uploadVehiclePositions}\}, \mathrm{FogLayer},\\
&\quad \{\mathrm{getVehiclePositions}, \mathrm{setPosition}\})\} \cup\\
&\quad \{(\{\}, \mathrm{CloudLayer}, \{\mathrm{uploadVehiclePositions}^*\}),\\
&\quad (\{\mathrm{uploadVehiclePositions}^*\}, \mathrm{FogLayer2}, \{\mathrm{uploadVehiclePositions}\}),\\
&\quad (\{\mathrm{uploadVehiclePositions}\}, \mathrm{FogLayer1},\\
&\quad \{\mathrm{getVehiclePositions}, \mathrm{setPosition}\})\}\\
&\mathrm{CommunicationSet} = \mathrm{CommunicationSet}_{old} \setminus\\
&\qquad\qquad \{(\mathrm{MunicipalityServer}, \mathrm{Ethernet}, \mathrm{TrafficLight1})^{\leftrightarrow},\\
&\qquad\qquad (\mathrm{MunicipalityServer}, \mathrm{Ethernet}, \mathrm{TrafficLight2})^{\leftrightarrow}\} \cup\\
&\qquad\qquad \{(\mathrm{MunicipalityServer}, 3G, \mathrm{CityDistrict})^{\leftrightarrow},\\
&\qquad\qquad (\mathrm{CityDistrict}, \mathrm{Ethernet}, \mathrm{TrafficLight1})^{\leftrightarrow},\\
&\qquad\qquad (\mathrm{CityDistrict}, \mathrm{Ethernet}, \mathrm{TrafficLight2})^{\leftrightarrow}\}
\end{aligned}$$


Cloud Layer
«Cloud Device» :MunicipalityServer
Ethernet
ALP
Fog Layer 2
«Fog Component» :CityDistrict
Ethernet
Ethernet
Fog Layer 1
«Fog Component» Traffic Light 1: TrafficLight
«Fog Component» Traffic Light 2: TrafficLight
WIFI
Edge Layer
«Edge Device» Vehicle 1: Vehicle
«Edge Device» Vehicle 2: Vehicle
«Edge Device» Vehicle 3: Vehicle
«Edge Device» Vehicle 4: Vehicle
«Edge Device» Vehicle 5: Vehicle
«Edge Device» Vehicle 6: Vehicle
Cloud
Fog
Edge

(a) View$_1$: The first 3-layered View of the Fog Architecture. The Abstraction Level is set to the third layer which relates to the `Fog Layer 2`.


ALP
Cloud Layer
«Cloud Device» :MunicipalityServer
Ethernet
Fog Layer 2
«Fog Component» :CityDistrict
Ethernet
Ethernet
Fog Layer 1
«Fog Component» Traffic Light 1: TrafficLight
«Fog Component» Traffic Light 2: TrafficLight
WIFI
Edge Layer
«Edge Device» Vehicle 1: Vehicle
«Edge Device» Vehicle 2: Vehicle
«Edge Device» Vehicle 3: Vehicle
«Edge Device» Vehicle 4: Vehicle
«Edge Device» Vehicle 5: Vehicle
«Edge Device» Vehicle 6: Vehicle
Cloud
Fog
Edge

(b) View$_2$: The second 3-layered View of the Fog Architecture. The Abstraction Level is set to the fourth layer which relates to the `Fog Layer 3`.

Figure 6.33: The HW/SW mapping of the Fog.BOI showing the different 3-layered Views (UML Deployment Diagram). For readability, we only show components and the used communication channels. It shows a 4-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer 2*, *Fog Layer 1*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers.

The resulting 4-layered Fog Architecture can be separated in two 3-layered Views as shown in Figure 6.33. These Views are defined by Equation 71. View$_1$ is set to Abstraction Level 2 which refers to *Fog Layer 2* and includes three layers: *Fog Layer 2*, *Fog Layer 1*, and the *Edge Layer*. It describes the functionality which was encompassed by the initial setup before an additional layer was added.

The second View, View$_2$, focuses on Abstraction Level 3 which refers to the *Cloud Layer*. It includes the *Cloud Layer*, *Fog Layer 2*, and *Fog Layer 1*. This View addresses the functionality which is based on the aggregated data gathered by the introduced city districts. Thus, it can show differences between the districts in terms of traffic.

Equation 71: Fog.BOI: Views

$$View(ALP, n) := \bigcup_{i=0}^{n-1} Layer(ALP - i)$$

$$\begin{aligned} View_1(2,3) :&= \bigcup_{i=0}^{n-1} Layer(ALP - i) = Layer(2) \cup Layer(1) \cup Layer(0) \\ &= \{FogLayer_2, FogLayer_1, EdgeLayer\} \\ View_2(3,3) :&= \bigcup_{i=0}^{n-1} Layer(ALP - i) = Layer(3) \cup Layer(2) \cup Layer(1) \\ &= \{CloudLayer, FogLayer_2, FogLayer1\} \end{aligned}$$

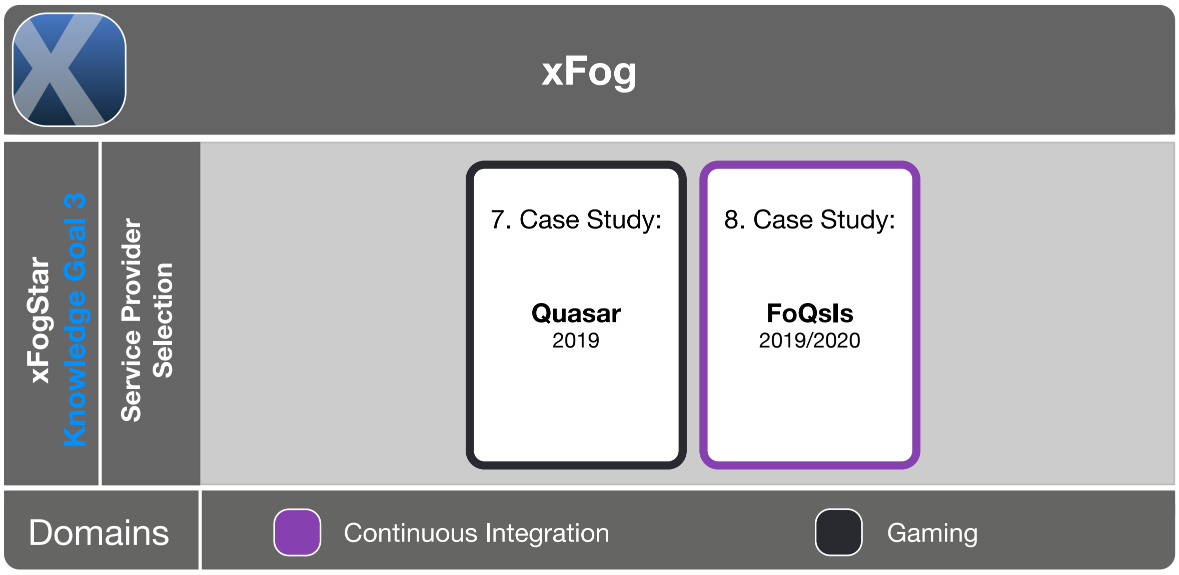


Figure 6.34: Excerpt of the validation design overview shown in Figure 6.1 for the Service Provider Selection.

6.2.3 Service Provider Selection

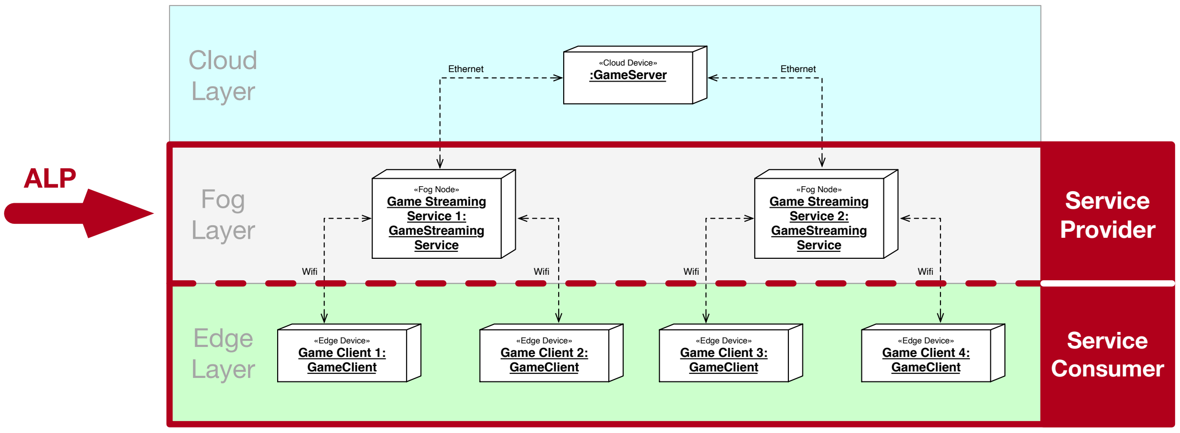
In this section, we look at the results of the validation for the Service Provider Selection highlighted by Figure 6.34. We use the case studies *Quasar* and *FoQsIs*, and show how the xFogStar workflow, as introduced in Chapter 5, was instantiated and how service providers are selected in the respective domains.

Thus, we present which unavailability strategies are selected, which limits are set, which comparability strategy we decided to use, which ordering strategy we applied, and which importance was assigned for the QoS parameters. This ultimately resulted in an ordered list of service providers from which we selected the first one in the list, respectively, the best fitting one.

Quasar

The first case study in the service provider selection validation is *Quasar*. Quasar used the Fog Architecture setup as shown in Figure 6.35.

It represented a 3-layered Fog Architecture with a `Game Server` on the *Cloud Layer*, two `Game Streaming Services` on the *Fog Layer*, and four `Game Clients` on the *Edge Layer*. As described in Section 6.1.7, the `Game Server` ran the game instance that was used by the remote `Game Clients`. To optimize connectivity, `Game Streaming Services` on the Fog Layer buffered game states that the client might request.



Cloud Layer
«Cloud Device» :GameServer
Ethernet
Ethernet
ALP
Fog Layer
«Fog Node» Game Streaming Service 1: GameStreaming Service
«Fog Node» Game Streaming Service 2: GameStreaming Service
Wifi
Wifi
Wifi
Wifi
Service Provider
Edge Layer
«Edge Device» Game Client 1: GameClient
«Edge Device» Game Client 2: GameClient
«Edge Device» Game Client 3: GameClient
«Edge Device» Game Client 4: GameClient
Service Consumer

Figure 6.35: The HW/SW mapping of the Quasar setup (UML Deployment Diagram). For readability, we only show components and the used communication channels. It shows a 3-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers. For the service provider selection, we use a 2-layered View with the ALP set to 1.

In this case study, we focus on the selection of the best fitting `Game Streaming Service`. Thus, according to the View concept introduced in Section 4.2.2, we used a 2-layered View which highlights the service provider and service consumer relation. The View is defined in Equation 72.

Equation 72: Quasar: View

$$\mathsf{View}(\mathsf{ALP}, n) := \bigcup_{i=0}^{n-1} \mathsf{Layer}(\mathsf{ALP} - i)$$

$$\begin{aligned}\mathsf{View}(1,2) &= \bigcup_{i=0}^{n-1} \mathsf{Layer}(\mathsf{ALP} - i) = \mathsf{Layer}(1) \cup \mathsf{Layer}(0) \\ &= \{\mathsf{FogLayer}, \mathsf{EdgeLayer}\}\end{aligned}$$

As the presented Fog Components are part of the Fog Architecture, we take a look at a new `Game Client 5` that searches for a `Game Streaming Service` to connect to. Although this component describes a dynamic addition, we focus on the instantiation of the xFogStar workflow, and thus we do not include the corresponding sets in this section. The sets can be found in the appendix in Section B.8.

The difference between `Game Client 5` and the other `Game Clients` was its Fog Visibility as shown in Figure 6.36.

While the other `Game Clients`' Fog Visibilities only included one `Game Streaming Service` each, `Game Client 5`'s Fog Visibility included both `Game Streaming Services`. Its Fog Visibility is highlighted in green including a dashed border. The Fog Visibilities of `Game Client 1-4` are shown as partly transparent blue circles around their labels. The red circles represent the Fog Visibilities of the game streaming services, while the yellow circle shows the Fog Visibility of the `Game Server` itself. According to those Fog Visibilities, `Game Client 5` could select between both game streaming services.

To select the best fitting service according to the service consumers needs, we instantiated xFogStar as introduced in Chapter 5. Equation 73 shows the QoS parameters we chose for the service provider comparison.

Equation 73: Quasar: Parameter Selection

$$\overrightarrow{QoS} = \begin{pmatrix} ExecutionTime(ET) \\ Cost(C) \\ Energy(E) \\ Storage(S) \\ Availability(A) \\ CommunicationTime(CT) \\ Bandwidth(B) \end{pmatrix}$$

In the Quasar app, Figure 6.37 shows the *Service Discovery* screen on the left which allowed to discover game services and to access its details as shown on the right screen. The details included all requested QoS parameters and the possibility to select this service provider.

As shown by Figure 6.35 and Figure 6.36, our Quasar setup included two game streaming services. Table 6.4 shows the QoS parameters for both. The game streaming services were similar, but mainly differed in the execution time, communication time, and their costs.

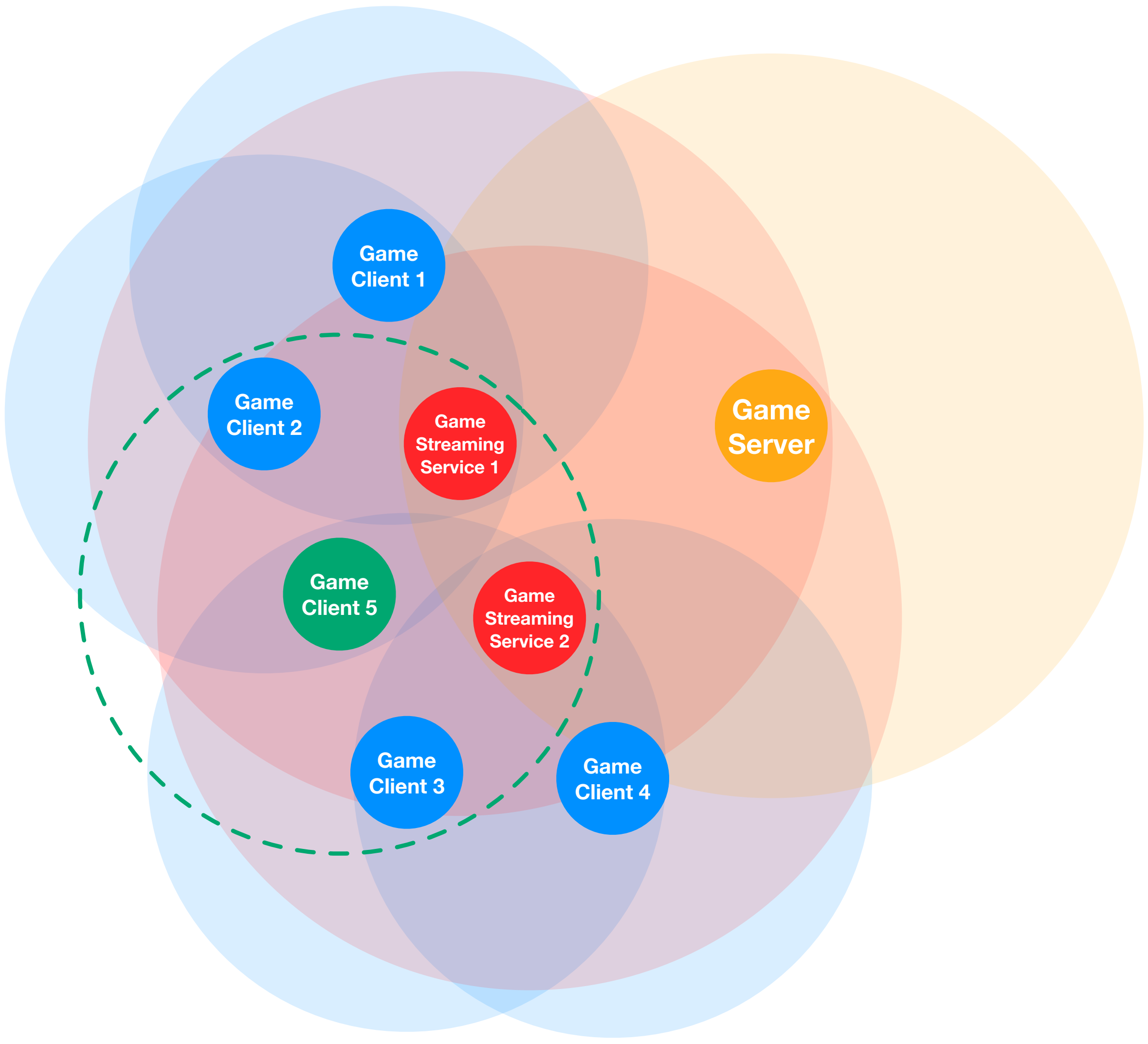
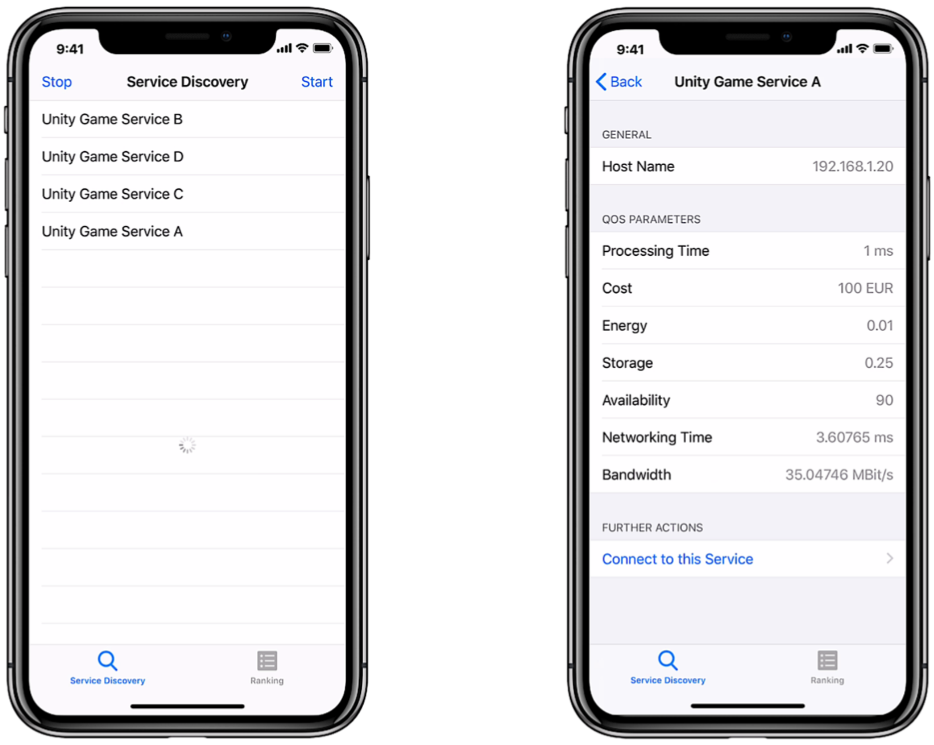


Figure 6.36: The Fog Visibilities of the different Fog Components included within the Quasar Fog Architecture. The *blue* blue circles represent Game Clients that are already connected to the Fog Architecture with only one Fog Component of the Fog Layer within their Fog Visibility. The *red* circles represent the corresponding Fog Nodes and the *yellow* circle the `Game Server`. The *green* circle shows a Fog Component which wants to connect to the Fog Architecture, but has two potential Fog Nodes within its Fog Visibility.

Table 6.4: Quasar: Parameter Values.

QoS Parameter	*Game Streaming Service 1*	*Game Streaming Service 2*
Execution Time (ET)	10ms	15ms
Cost (C)	$10	$1
Energy (E)	0.01	0.01
Storage (S)	0.01	0.01
Availability (A)	99.0	99.0
Communication Time (CT)	10ms	15ms
Bandwidth (B)	10 Mbit/s	8 Mbit/s


9:41

Stop Service Discovery Start

Unity Game Service B

Unity Game Service D

Unity Game Service C

Unity Game Service A

Service Discovery Ranking

9:41

Back Unity Game Service A

GENERAL

Host Name	192.168.1.20

QOS PARAMETERS

Processing Time	1 ms
Cost	100 EUR
Energy	0.01
Storage	0.25
Availability	90
Networking Time	3.60765 ms
Bandwidth	35.04746 MBit/s

FURTHER ACTIONS

Connect to this Service

Service Discovery Ranking

Figure 6.37: Quasar App in the *Service Discovery* Tab. The screen on the left shows the screen to start and stop the discovery process and the resulting list of found service providers. The second screen shows a detailed overview of the currently selected service provider.

Equation 74 simplifies the values of Table 6.4 to a QoS matrix, which is needed for the next steps, with the QoS vector of `Game Streaming Service 1` in the first column and the QoS Vector of `Game Streaming Service 2` in the second column. For readability, we keep the abbreviated QoS parameters on the left of the matrix.

Equation 74: Quasar: QoS Matrix

$$\begin{array}{c} ET \\ C \\ E \\ S \\ A \\ CT \\ B \end{array} \begin{pmatrix} 10 & 15 \\ 10 & 1 \\ 0.01 & 0.01 \\ 0.01 & 0.01 \\ 99.0 & 99.0 \\ 10 & 15 \\ 10 & 8 \end{pmatrix}$$

According to the xFogStar workflow, the first step is the selection of an unavailability strategy. As the QoS matrix is already fully populated with values, we can skip this step and proceed to the definition of parameter limits. We do not set any limits, because we only have two service providers, but we will show how limits are defined and displayed in the Quasar app at the end of this section. For the third step of the workflow, the selection of the comparability strategy, we select the percentage comparability as shown in Equation 75. Thus, for every row, we sum up the values and divide each individual value by this sum to receive its share.

Equation 75: Quasar: Percentage Comparability

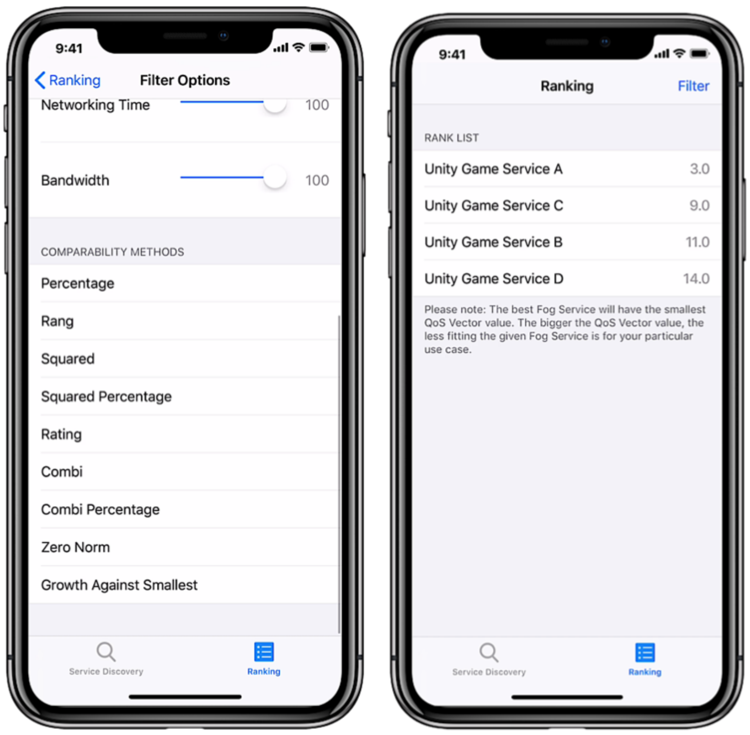
$$\begin{array}{c} \text{ET} \\ \text{C} \\ \text{E} \\ \text{S} \\ \text{A} \\ \text{CT} \\ \text{B} \end{array} \begin{pmatrix} 10 & 15 \\ 10 & 1 \\ 0.01 & 0.01 \\ 0.01 & 0.01 \\ 99.0 & 99.0 \\ 10 & 15 \\ 10 & 8 \end{pmatrix} \rightarrow \begin{pmatrix} 10/25 & 15/25 \\ 10/11 & 1/11 \\ 0.01/0.02 & 0.01/0.02 \\ 0.01/0.02 & 0.01/0.02 \\ 99.0/198 & 99.0/198 \\ 10/25 & 15/25 \\ 10/18 & 8/18 \end{pmatrix} \rightarrow \begin{pmatrix} 0.40 & 0.60 \\ 0.91 & 0.09 \\ 0.50 & 0.50 \\ 0.50 & 0.50 \\ 0.50 & 0.50 \\ 0.40 & 0.60 \\ 0.56 & 0.44 \end{pmatrix}$$

After applying the comparability strategy, we have to address the order of the parameters. Most of the selected QoS parameters require lower values, but the *Availability* and *Bandwidth* are the better the higher they are. Based on the percentage values we created during the comparability strategy, we can invert these QoS parameters as shown in Equation 76.

Equation 76: Quasar: Ordering

$$\begin{array}{c} \text{ET} \\ \text{C} \\ \text{E} \\ \text{S} \\ \text{A} \\ \text{CT} \\ \text{B} \end{array} \begin{pmatrix} 0.40 & 0.60 \\ 0.91 & 0.09 \\ 0.50 & 0.50 \\ 0.50 & 0.50 \\ 0.50 & 0.50 \\ 0.40 & 0.60 \\ 0.56 & 0.44 \end{pmatrix} \rightarrow \begin{pmatrix} 0.40 & 0.60 \\ 0.91 & 0.09 \\ 0.50 & 0.50 \\ 0.50 & 0.50 \\ 1 - 0.50 & 1 - 0.50 \\ 0.40 & 0.60 \\ 1 - 0.56 & 1 - 0.44 \end{pmatrix} \rightarrow \begin{pmatrix} 0.40 & 0.60 \\ 0.91 & 0.09 \\ 0.50 & 0.50 \\ 0.50 & 0.50 \\ 0.50 & 0.50 \\ 0.40 & 0.60 \\ 0.44 & 0.56 \end{pmatrix}$$

Finally, we can define the importance of each QoS parameter. We set every parameter to the same importance, and thus skip this step. Figure 6.38 shows how the ranking process is represented in the Quasar app. On the left screen, the service consumer can define the parameter importance at the top of the screen and select the comparability strategy at the bottom of the screen. Afterwards, the resulting ordered list of service providers with their respective value is shown as depicted on the right screen.


9:41

Ranking | Filter Options

Networking Time	100
Bandwidth	100

COMPARABILITY METHODS

- Percentage
- Rang
- Squared
- Squared Percentage
- Rating
- Combi
- Combi Percentage
- Zero Norm
- Growth Against Smallest

9:41

Ranking | Filter

RANK LIST

Unity Game Service A	3.0
Unity Game Service C	9.0
Unity Game Service B	11.0
Unity Game Service D	14.0

Please note: The best Fog Service will have the smallest QoS Vector value. The bigger the QoS Vector value, the less fitting the given Fog Service is for your particular use case.

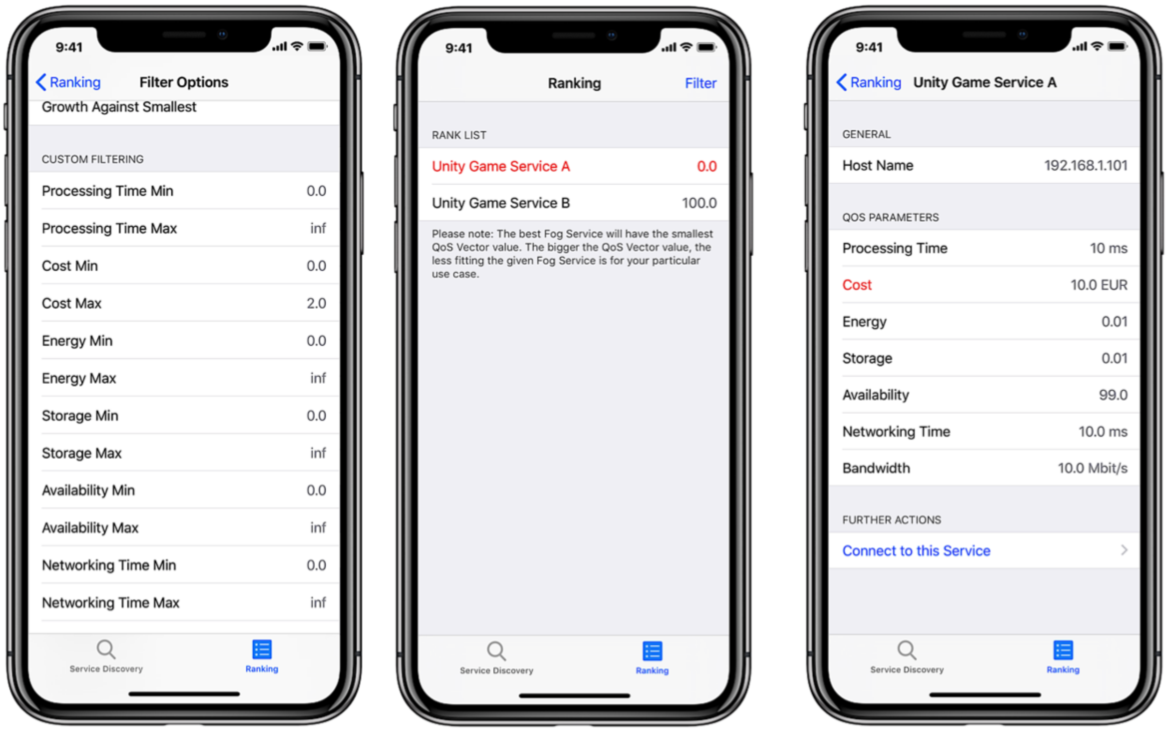
Figure 6.38: Quasar App in the *Ranking* Tab. The first screen shows the different implemented comparability strategies which can be selected. The second screen shows the ordered list for the available service providers with their according value based on the selected strategy.

The values are calculated based on Definition 38. The values for our Quasar case study are shown in Equation 77. Accordingly, `Game Streaming Service 2` is selected as the streaming service for Fog Component `Game Client 5`.

Equation 77: Quasar: Ordered Service Provider List

$$\begin{array}{c} \text{ET} \\ \text{C} \\ \text{E} \\ \text{S} \\ \text{A} \\ \text{CT} \\ \text{B} \end{array} \begin{pmatrix} 0.40 & 0.60 \\ 0.91 & 0.09 \\ 0.50 & 0.50 \\ 0.50 & 0.50 \\ 0.50 & 0.50 \\ 0.40 & 0.60 \\ 0.44 & 0.56 \end{pmatrix} \rightarrow \begin{array}{cc} \text{QoS}_1 & \text{QoS}_2 \\ \end{array} \begin{pmatrix} 3.65 & 3.35 \end{pmatrix}$$

Figure 6.39 shows how limits for QoS parameters could be set within the Quasar app. These limits include maximum and minimum values. If we set the maximum costs to 2.0, every service provider that exceeds this limit is highlighted in red in the final ranking. In the details, service consumers can see which limits were exceeded and decide if they want to use this service provider anyways or if they want to select a worse fit.



9:41

Ranking | Filter Options

Growth Against Smallest

CUSTOM FILTERING

Processing Time Min	0.0
Processing Time Max	inf
Cost Min	0.0
Cost Max	2.0
Energy Min	0.0
Energy Max	inf
Storage Min	0.0
Storage Max	inf
Availability Min	0.0
Availability Max	inf
Networking Time Min	0.0
Networking Time Max	inf

Service Discovery | Ranking

9:41

Ranking | Filter

RANK LIST

Unity Game Service A	0.0
Unity Game Service B	100.0

Please note: The best Fog Service will have the smallest QoS Vector value. The bigger the QoS Vector value, the less fitting the given Fog Service is for your particular use case.

Service Discovery | Ranking

9:41

Ranking | Unity Game Service A

GENERAL

Host Name	192.168.1.101

QOS PARAMETERS

Processing Time	10 ms
Cost	10.0 EUR
Energy	0.01
Storage	0.01
Availability	99.0
Networking Time	10.0 ms
Bandwidth	10.0 Mbit/s

FURTHER ACTIONS

Connect to this Service

Service Discovery | Ranking

Figure 6.39: Limit definition in the Quasar App. On the first screen, the service consumer can define different max and min values for the given QoS parameters. The second screen shows the ranking after a comparability was selected. The *red* highlighting shows service providers that exceeded one parameter limit. On the third screen, a detailed overview, the corresponding parameter is highlighted.

FoQsIs

The second case study for the validation of the *Service Provider Selection* is *FoQsIs* with Figure 6.40 showing its HW/SW mapping.

It shows a 3-layered Fog Architecture: On the *Cloud Layer* it contained one Fog Component called `StatsServer` which gathered the statistics of the `Integration Services`, of which five were placed on the *Fog Layer*. They were used to provide continuous integration services to the `FoQsIs Clients` which were placed on the *Edge Layer*.

In accordance to the Quasar case study, FoQsIs focuses on the selection of the best fitting integration service provider. As shown in Chapter 5, this required the introduction of a 2-layered *View* highlighting service consumers and possible service providers as defined in Equation 78.

Equation 78: FoQsIs: View

$$View(ALP, n) := \bigcup_{i=0}^{n-1} Layer(ALP - i)$$

$$View(1, 2) = \bigcup_{i=0}^{n-1} Layer(ALP - i) = Layer(1) \cup Layer(0) = \{FogLayer, EdgeLayer\}$$

Since Figure 6.40 shows the current setup with already connected `FoQsIs Clients`, in the following, we investigate `FoQsIs Client 3` which wanted to use the services provided by the integration service providers. The Fog Visibilities of the `Integration Services` and the Fog Visibility of the `FoQsIs Client 3` are shown in Figure 6.41. Based on its Fog Horizon, `FoQsIs Client 3` could select between all five `Integration Services`.

We instantiated xFogStar to select the best fitting service provider in accordance to the needs of `FoQsIs Client 3`. Equation 79 shows the parameter selection for the FoQsIs case study among which the *Delay* can be mapped to the *Time* as introduced in Section 5.1.1.

Equation 79: FoQsIs: Parameter Selection

$$\overrightarrow{QoS} = \begin{pmatrix} Delay(D) \\ Cost(C) \\ Bandwidth(B) \\ Availability(A) \end{pmatrix}$$

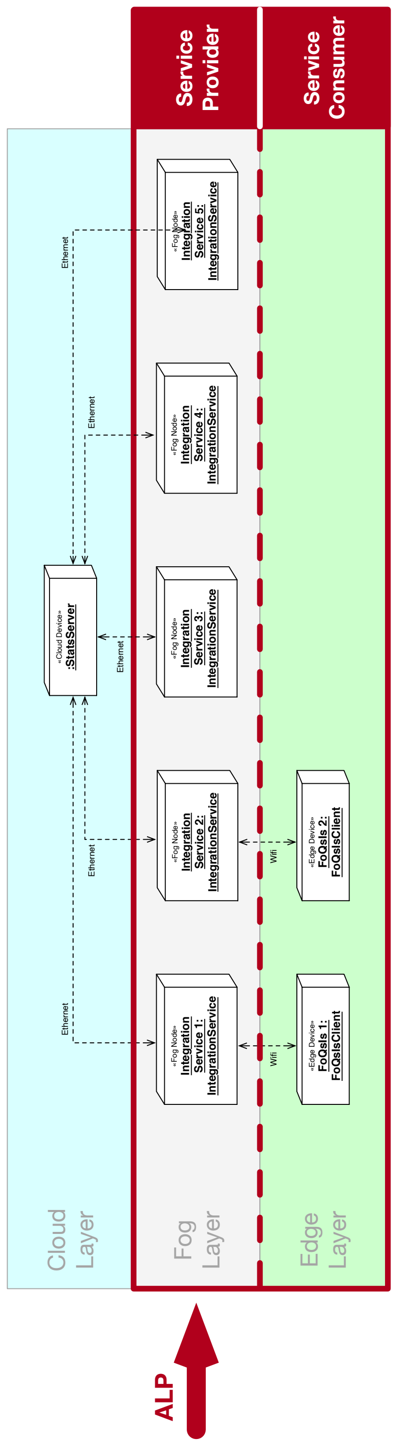

Service Provider
Service Consumer
Cloud Layer
Fog Layer
Edge Layer
«Cloud Device» :StatsServer
«Fog Node» Integration Service 1: IntegrationService
«Fog Node» Integration Service 2: IntegrationService
«Fog Node» Integration Service 3: IntegrationService
«Fog Node» Integration Service 4: IntegrationService
«Fog Node» Integration Service 5: IntegrationService
«Edge Device» FoQsIs 1: FoQsIsClient
«Edge Device» FoQsIs 2: FoQsIsClient
Ethernet
Wifi
ALP

Figure 6.40: The HW/SW mapping of the FoQsIs setup (UML Deployment Diagram). For readability, we only show components and the used communication channels. It shows a 3-layered Fog Architecture containing a *Cloud Layer*, *Fog Layer*, and the *Edge Layer*, which are all deployed on different hardware components, thus, also representing tiers. For the service provider selection, we use a 2-layered View with the ALP set to 1.

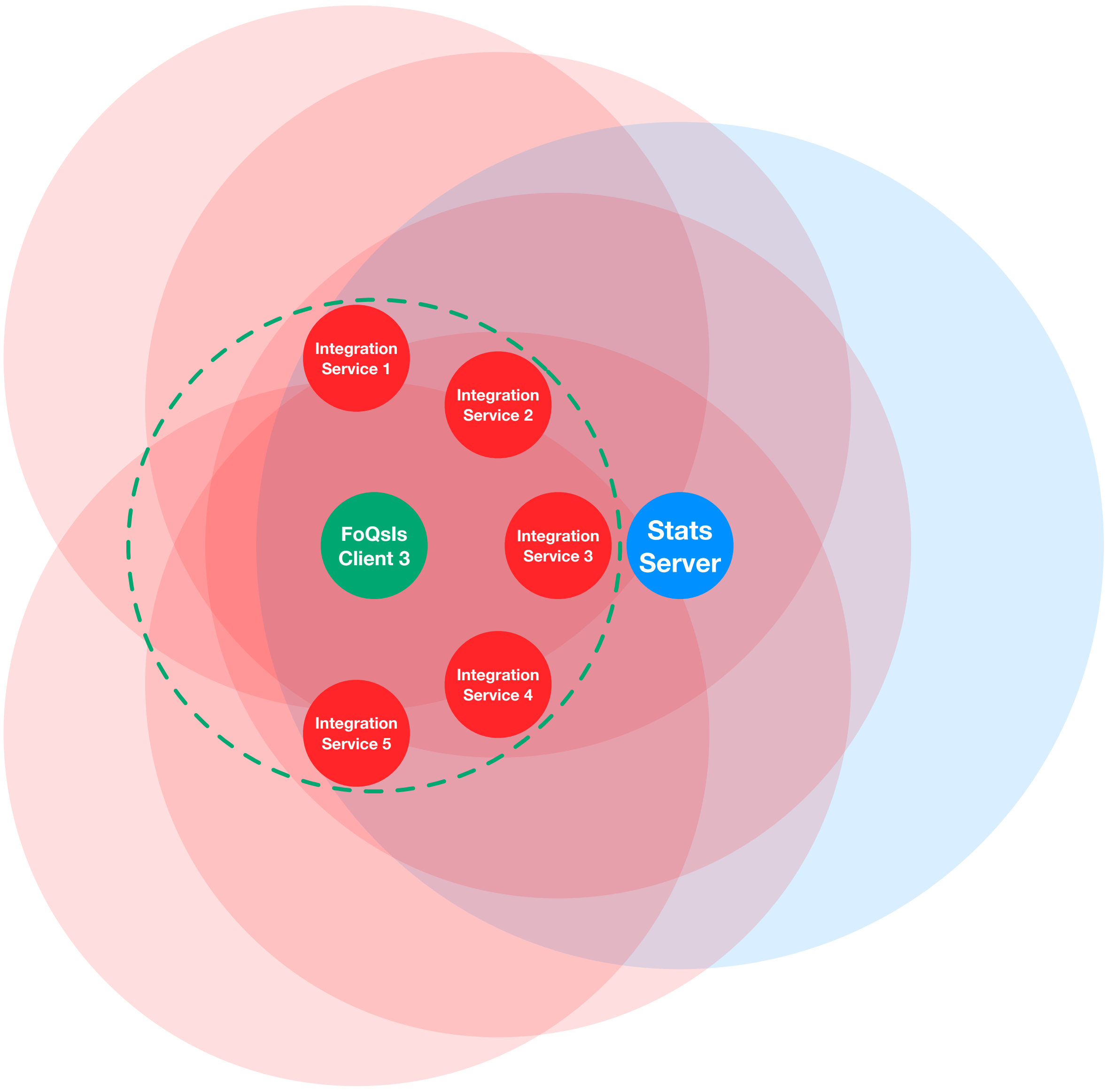


Figure 6.41: The Fog Visibilities of the different Fog Components included within the FoQsIs Fog Architecture. For readability, we exclude both `FoQsIs Clients` that are already connected. The *red* circles represent the `Integration Services` and the *blue* circle the `Stats Server`. The *green* circle shows a Fog Component which wants to connect to the Fog Architecture, but has five potential Fog Nodes within its Fog Visibility.

Table 6.5: FoQsIs: Parameter Values.

QoS Parameter	*Integration Service 1*	*Integration Service 2*	*Integration Service 3*	*Integration Service 4*	*Integration Service 5*
Delay (D)	1.5s	0.9s	4.8s	0.2s	2.8s
Cost (C)	$5	$1	$1	$19	$9
Bandwidth (B)	175 Mbit/s	375 Mbit/s	25 Mbit/s	975 Mbit/s	102 Mbit/s
Availability (A)	99.5%	95.3%	91.5%	99.7%	97.5%

The parameter values for the five `Integration Services` are presented in Table 6.5. In comparison to the Quasar case study, for these service providers, it is not inherently obvious which is the best fitting one. It is highly dependent on the priorities given by the service consumer `FoQsIs Client 3`.

We transform the table to a matrix representation as shown in Equation 80 to make it easier to work with. Each service provider's QoS parameters are listed in one column ordered by the service provider's number.

Equation 80: FoQsIs: QoS Matrix

$$\begin{array}{c} D \\ C \\ B \\ A \end{array} \begin{pmatrix} 1.5 & 0.9 & 4.8 & 0.2 & 2.8 \\ 5 & 1 & 1 & 19 & 9 \\ 175 & 375 & 25 & 975 & 102 \\ 99.5 & 95.3 & 91.5 & 99.7 & 97.5 \end{pmatrix}$$

As the QoS matrix for this example of the FoQsIs case study was fully populated, we could skip the first step of the xFogStar workflow which required the selection of an unavailability strategy.

Equation 81 shows the defined limits: We did not set any limits for the *Delay*, *Costs*, or *Bandwidth*, but required an *Availability* of at least 95%.

Equation 81: FoQsIs: Limits

$$\begin{array}{c} D \\ C \\ B \\ A \end{array} \begin{pmatrix} [0;\infty] \\ [0;\infty] \\ [0;\infty] \\ [95;100] \end{pmatrix}$$

Only one service provider, `Integration Service 3` could not ensure this *Availability* and was therefore removed from the further calculations and would not be considered as a suitable service provider.

For the comparability strategy, we selected a strategy that we call piecewise linear comparability. For each parameter, it defines a piecewise function consisting of four parts as shown in Definition 39. For parameters that prefer high values, this relates to a continuous increasing function and for parameters that prefer low values, this relates to a continuous decreasing function.

Definition 39: Piecewise Linear Comparability

For parameters preferring high values :

$$f(x) := \begin{cases} 0 & x \leqslant 0 \\ \frac{1}{base}\,x & 0 \leqslant x \leqslant base \\ -\frac{1}{base-max}\,x + 2 + \frac{max}{base-max} & base \leqslant x \leqslant max \\ 2 & max \leqslant x \end{cases}$$

For parameters preferring low values :

$$f(x) := \begin{cases} 2 & x \leqslant 0 \\ -\frac{1}{base}\,x + 2 & 0 \leqslant x \leqslant base \\ -\frac{1}{max-base}\,x + 1 + \frac{base}{max-base} & base \leqslant x \leqslant max \\ 0 & max \leqslant x \end{cases}$$

To create the according functions, we had to define two values: *base* and *max*. *Base* relates to the expected value for the parameter and *max* relates to the value above which no improvement (in case of parameters that prefer high values) and no worsening (in case of parameters that prefer low values) should occur. The *base* value always gets assigned a value of 1. For the increasing function, the lowest value is set to 0 at 0 and the highest value is set to 2 at max. In contrast, for the decreasing function, the lowest value is set to 2 at 0 and the highest value is set to 0 at max. The resulting three points are then linear connected.

We defined the base and max values for our parameters as shown in Equation 82.

Equation 82: FoQsIs: Base and Max Values

$$\begin{aligned}
&baseDelay = 2\\
&baseCost = 1\\
&baseBandwidth = 20\\
&baseAvailability = 99\\
\\
&maxDelay = 100\\
&maxCost = 25\\
&maxBandwidth = 500\\
&maxAvailability = 100
\end{aligned}$$

Using these base and max values, Equation 83 shows the according functions.

Equation 83: FoQsIs: Piecewise Linear Comparability

Delay :

$$f(x) := \begin{cases} 2 & x \leqslant 0 \\ -\frac{1}{2}x + 2 & 0 \leqslant x \leqslant 2 \\ -\frac{1}{98}x + 1 + \frac{1}{49} & 2 \leqslant x \leqslant 100 \\ 0 & 100 \leqslant x \end{cases}$$

Cost :

$$f(x) := \begin{cases} 2 & x \leqslant 0 \\ -x + 2 & 0 \leqslant x \leqslant 1 \\ -\frac{1}{24}x + \frac{25}{24} & 1 \leqslant x \leqslant 25 \\ 0 & 25 \leqslant x \end{cases}$$

Bandwidth :

$$f(x) := \begin{cases} 0 & x \leqslant 0 \\ \frac{1}{20}x & 0 \leqslant x \leqslant 20 \\ \frac{1}{480}x + \frac{23}{24} & 20 \leqslant x \leqslant 500 \\ 2 & 500 \leqslant x \end{cases}$$

Availability :

$$f(x) := \begin{cases} 0 & x \leqslant 0 \\ \frac{1}{99}x & 0 \leqslant x \leqslant 99 \\ x - 98 & 99 \leqslant x \leqslant 100 \\ 2 & 100 \leqslant x \end{cases}$$

Equation 84 shows the resulting values for our quality of service vector based on the functions.

Equation 84: FoQsIs: QoS Matrix

$$\begin{array}{c} D \\ C \\ B \\ A \end{array} \begin{pmatrix} 1.5 & 0.9 & 4.8 & 0.2 & 2.8 \\ 5 & 1 & 1 & 19 & 9 \\ 175 & 375 & 25 & 975 & 102 \\ 99.5 & 95.3 & 91.5 & 99.7 & 97.5 \end{pmatrix} \longrightarrow \begin{pmatrix} 1.25 & 1.55 & 1.90 & 0.99 \\ 0.83 & 1.00 & 0.25 & 0.67 \\ 1.32 & 1.74 & 2.00 & 1.17 \\ 1.50 & 0.96 & 1.70 & 0.98 \end{pmatrix}$$

The next step of the xFogStar workflow is the selection of the ordering strategy. This ordering was addressed using different definitions of continuous, piecewise functions depending on the ordering of parameters.

Before we could compare the service providers, we had to set the parameters importance by defining weights. These are shown in Equation 85 with the highest importance assigned to the *Availability*.

Equation 85: FoQsIs: Weights

$$\begin{array}{c} D \\ C \\ B \\ A \end{array} \begin{pmatrix} 30\% \\ 15\% \\ 60\% \\ 100\% \end{pmatrix}$$

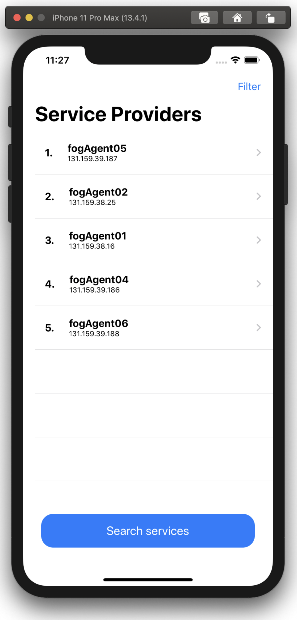
Finally, we could add up the values for each service provider to evaluate the best fit for our service consumer *FoQsIs Client 3*.

Equation 86: FoQsIs: Ordered Service Provider List

$$\begin{array}{c} D \\ C \\ B \\ A \end{array} \begin{pmatrix} 1.25 & 1.55 & 1.90 & 0.99 \\ 0.83 & 1.00 & 0.25 & 0.67 \\ 1.32 & 1.74 & 2.00 & 1.17 \\ 1.50 & 0.96 & 1.70 & 0.98 \end{pmatrix} \rightarrow \begin{array}{cccc} QoS_1 & QoS_2 & QoS_4 & QoS_5 \\ \end{array} \begin{pmatrix} 2.79 & 2.62 & 3.51 & 2.08 \end{pmatrix}$$

The representation of the xFogStar workflow in the FoQsIs application is shown in Figure 6.42. On the first screen, the discovery process found the five `Integration Services` which are represented in the HW/SW mapping in Figure 6.40. They were called *fogAgent01*, *fogAgent02*, *fogAgent04*, *fogAgent05*, and *fogAgent06*. The second screen shows the step of setting limits and defining weights for the parameters importance. After saving the given limits and weights, the third screen shows the ordered list of *fogAgents* with the best fitting service provider on top, excluding *fogAgent04* which did not ensure the required *Availability*.

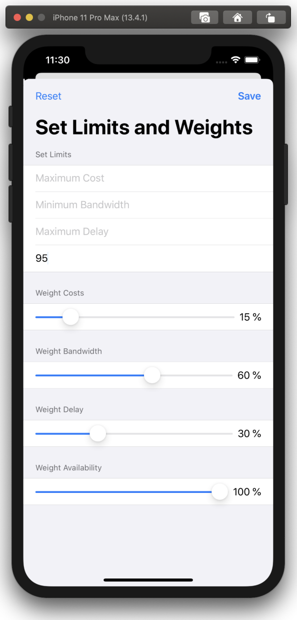
After the xFogStar workflow, the application user can select one of the service providers and establish a connection.



Service Providers

1. fogAgent05 131.159.39.187
2. fogAgent02 131.159.38.25
3. fogAgent01 131.159.38.16
4. fogAgent04 131.159.39.186
5. fogAgent06 131.159.39.188

(a) Service Discovery (1)



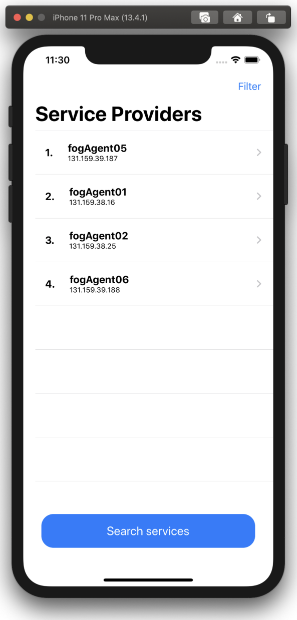
Set Limits and Weights

Set Limits

Maximum Cost
Minimum Bandwidth
Maximum Delay
95

Weight Costs: 15 %
Weight Bandwidth: 60 %
Weight Delay: 30 %
Weight Availability: 100 %

(b) Limits & Weights (2)



Service Providers

1. fogAgent05 131.159.39.187
2. fogAgent01 131.159.38.16
3. fogAgent02 131.159.38.25
4. fogAgent06 131.159.39.188

(c) Service Provider List (3)

Figure 6.42: An overview of the *FoQsIs Client* implementation. The application provides the possibility to discover service providers for a given service (1), assign limits and weights (2), and finally returns an ordered list of service providers with the best fit on top and excluding providers that did not match the limits (3).

6.3 Discussion

In this section, we provide interpretations for the results of the three validations in form of multiple case studies as shown in Section 6.2. In Section 6.3.2, we investigate threats to the validity of the validations. For the interpretations, we distinguish between the individual validations, while for the threats to validity, we combine the first two validations, because they rely on the same validation technique.

6.3.1 Interpretation

The interpretation of each validation addresses at least two aspects: the *Expected Results versus Provided Results* and the *Ease of Applicability*. The first aspect describes the differences in the results between the definitions based on Fog Computing and the definitions introduced by the xFog framework. In *Ease of Applicability*, we address the effort it takes to apply the concepts and issues that arose during the implementation of the case studies.

Independent of the individual validations, all case studies are based on the basic concepts of xFog (xFogCore) which describe the component set and the communication set of the Fog Architecture. As we could provide results for all of the case studies and the related concepts, it supports the assumption that the foundations of xFog provide the needed basis for the investigated Fog Architectures for the more advanced concepts: *Dynamic Fog Components*, *Scalable Fog Architectures*, and *Service Provider Selection*.

Further investigations for the foundations of xFog could include simulations of Fog Architectures in particular if algorithms are developed that automate the process of the evaluation.

Dynamic Fog Components

The interpretation of the results of the *Dynamic Fog Components* includes three aspects: *Expected Results versus Provided Results*, *Overhead*, and *Security*. While the first two match the overall aspects, during implementation of the case studies, the security aspect came up several times.

Expected Results versus Provided Results In order to differentiate between the results provided by xFogCore and xFogPlus, each cases study in the Dynamic Fog Components validation describes the initial setup of the case study as a HW/SW mapping based on a simplified UML component diagram. This allows us to put the results provided by the mathematical definitions of xFogCore and xFogPlus in relation to the findings we would expect based on the model approach used by a software architect.

The shown setups also include the component which should dynamically be added to the Fog Architecture and the placement within the architecture that is expected based on the components' context. While for the xFogPlus definitions the component needs to conform to certain criteria to be considered a Fog Component, the placement of the Fog Component based on the mathematical sets conforms to the expected placement within the Fog Architecture for all three case studies. This supports the idea that the xFog foundations describe the dynamic addition of a Fog Component to a Fog Architecture.

Overhead Due to the amount of mathematical concepts needed to create a consistent definition for a Fog Architecture and the dynamic addition of new components to the Fog Architecture, there is an overhead which is needed to apply the definitions to a Fog Architecture and to represent the addition of a new component. First, the process of adding a new component to an existing Fog Architecture requires the proof that the component complies to the Fog Component requirements. Additionally, the Fog Set, Communication Set, Service Set, Service Set Mapping, and the Layer Definitions need to be evaluated to address the impact of the new Fog Component on the existing Fog Architecture. As we defined the concepts but do not provide any algorithms for automation, yet, the overhead to evaluate those mathematical sets in comparison to the model based approach using component diagrams which is common in Software Engineering is not neglectable. For comparison, in the model based approach, it is sufficient to describe the new component in relation to the other components of the Fog Architecture.

Security To create the mathematical sets required for the addition of a new component and particularly the placement of the new component on the according layer, the Fog Architecture needs to be fully understood and accessible by the system architect. For example, to define the different layers of the Fog Architecture, all provided services need to be discoverable and documented. This means that even almost independent components that are only connected to the Cloud Device, e. g., shown by the PdMFrame case study, need to disclose their provided services and connected components.

Scalable Fog Architectures

This section presents the interpretation of the scalable Fog Architecture validation. We describe the *Expected Results versus Provided Results* and address the *Overhead*. Additionally, we investigate the *benefits* the View concept includes and the *impact* of the dynamic type change of Fog Components.

Expected Results versus Provided Results In accordance to the first validation, every case study in the scalable Fog Architectures validation first describes the initial setup using a HW/SW mapping described as a UML component diagram. As the goal of the case studies is the addition of a new layer, the description of the initial setup using xFogCore and xFogPlus focuses on the contained layers in this setup, the services, which describe them, as well as the connections between the layers. The second presented setup for each case study shows the UML component diagram displaying the expected result after the new layer was integrated. To achieve this using the introduced concepts, several sets need to be adjusted. These adjustments can be translated back to a Fog Architecture which represents the setup which described the expected results. This supports the assumption that the scalable concept of xFogPlus works as intended.

Overhead The scalable Fog Architecture concept contains an overhead. Also not presented in Section 6.2.2 due to a different point of focus, the full description of the initial setup contains the same overhead as for the dynamic Fog Components. If the setup is described using the presented concepts, the actual addition of a new layer is straightforward with each set only requiring minor adjustments.

Benefits of the View concept As part of the scalable Fog Architecture concept, we tested the applicability of the View concept as introduced in Section 4.2.2. As shown, the View definition allows an easy usage, only requiring a single set to be evaluated which is described by the Abstraction Level and the amount of requested layers. Also the benefits of these Views is rather limited in the presented case studies, as the complexity of the Fog Architecture is not high enough, the benefits already increase in the eHealth setup (Equation 6.2.2) with a total of five layers. Describing different Abstraction Levels is even more interesting if different stakeholders are involved in a setup which has a sufficient scale and extend.

Impact of Dynamic Type Changes The final point which is addressed in the case studies for scalable Fog Architectures is the dynamic type change that several Fog Components go through based on the different described Views. While this change might not have any immediate influence, it describes a fundamental problem in the concept of Fog Computing. Depending on the stakeholders' definitions of components, different component are considered Edge Devices, Fog Nodes, and Cloud Devices. Thus, a definition which uniquely defines those three types of components needs to be introduced, e. g., the description of these components based on the offered and consumed services as described by xFogPlus.

Service Provider Selection

This section presents the interpretation of the validation for the Service Provider Selection. We describe *Expected Results versus Provided Results*, *Ease of Applicability*, and the *Strategy Selections*.

Expected Results versus Provided Results Due to the difference in the case study design for the Service Provider Selection validation in terms of instantiations of the xFogStar workflow in contrast to the modeling approach of the first two validations, the description of the initial setup differs in the sense that we only describe a 2-layered View and the Fog Components' Fog Visibilities before adding a new component to the setup. Accordingly, we cannot base our expected results on these descriptions. The result that is expected is that the service consumer finds the best fitting service provider for a given service. In the case of Quasar with only two available service providers, this is rather easy as the second service provider is better in almost every parameter and that's the result that we get by using the xFogStar workflow. But the more QoS parameters are included and the more service providers, the harder it is to define which service provider is the "best fitting" service provider for a service consumer. Even worse, the service provider selection is closely linked to the selected strategies for the different parts of the workflow. For example, a comparability strategy which uses absolute values might rate a service provider unusable based on one single bad QoS parameter, also it might be the better service provider over all. Thus, we focused on the instantiation of the workflow and that the workflow provides the expected results for examples that are understandable instead of including all the different strategies.

Ease of Applicability As shown by both case studies, implementing the strategies and exposing them to the users which can select the strategies as well as preferences is more a topic for UI design than the ease of applicability of the workflow itself. The workflow implementation can be made as complex as desired using different strategies, but the approach itself is designed to be rather simplistic and result oriented.

Strategy Selections One of the most difficult decisions for the xFogStar workflow is the selection of the strategies for the different parts of the workflow, as the strategies heavily influence the selection process of the service provider. While our focus is the applicability of the workflow itself, we suggest additional laboratory experiments to evaluate which strategy performs the best and which other strategies should be investigated for the unavailability strategies, the comparability strategies, and the ordering strategies.

6.3.2 Threats to Validity

In this section we want to address the threats to validity that exist for the three presented validations. As the first two validations are based on the introduction of the concepts using modeling and the third validation being based on instantiating the xFogStar workflow, we distinguish the threats dependent on those two different types of validation. We focus on the four dimensions of validity as introduced by Runeson et al. [117]: construct validity, internal validity, external validity, and reliability.

Modeling Validations

First, we want to address the threats to validity for both validations that mainly relied on modeling, as both are based on the same validation method, and therefore pose similar threats.

Construct Validation The goal of both validations is to show that the introduced foundations of xFog can be applied to the selected architectures in the different domains. Based on the assumption that xFog is a generalized concept which can be applied to any architecture that is based on Fog Computing, the probability of the case study not meeting the results the researcher wants to study can be considered low as long as the described cases represent a Fog Architecture. As the case studies are designed to proof the applicability by evaluating the mathematical definitions, they also do not provide a wide variety of interpretability.

Internal Validation Due to the fact, that the researcher was involved in the development of the case studies, also mainly as an advisor, raises the chance that the case studies were tailored to the definitions of xFog, and thus the presented case studies do not reflect Fog Architectures which are representative for a wide variety of Fog Architectures. We tried to mediate this threat to the internal validity by selecting different domains and address different needs introduced by these domains with respect to Fog Computing. Additionally, we involved, depending on the case study, up to ten developers in the modeling and development of the systems as well as other researchers that guided the research process.

External Validation We report on three case studies each for both validations, the degree of generalizability is considered low. We tried to select different domains to ensure the applicability of the introduced concepts across independent domains but both, ARControl and Lassie, as well as DisCoFog 1 and Fog.BOI introduce an overlap of application domains. ARControl and Lassie are both placed within the smart environment domain with Lassie being additionally in the health domain.

As the health domain relies on similar concepts as the smart environment domain, this overlap could not be avoided with this selection of case studies. Nevertheless, we argue that the health domain exhibits unique requirements which distance it from the smart environment domain.

In comparison, DisCoFog 1 and Fog.BOI are both placed within the smart city domain which tries to better integrate smart vehicles in traffic guidance and optimization. While the case studies differed in the vehicle selection, the domains do not differ enough to be considered different domains.

To increase the degree of generalizability, additional case studies in different domains need to be conducted for both validations.

Reliability Due to the foundation of xFog being based on mathematical definitions, the reliability of both validations can be considered high. After implementing the systems in the different domains, the case studies applied the mathematical definitions as introduced in xFogCore and xFogPlus. These mathematical definitions do not offer any room for interpretations, and thus lead to the same results independent of the researcher analyzing the case studies results.

Instantiation Validation

In this section, we address the threats to validity for the instantiations of the xFogStar workflow validation.

Construct Validation We particularly selected the validation and implemented the case studies with the xFogStar workflow in mind. Thus, the risk of the case studies not representing the intended goals, can be considered low. Also the case studies were selected by the researcher, the implementations were created by students during the conduction of their theses. Thus, the risk is higher that the understanding of the workflow differed between the researchers intentions and the students' interpretations. We tried to address this issue by scheduling weekly meetings to discuss the students progress and answer any upcoming question. Additionally, they could get in contact with us at any time.

Internal Validation While conducting the case studies, there is a risk that third factors influenced the results of the validation of the workflow. Especially, the selection of the strategies for each step of the workflow poses such a threat. To mediate the threats to the internal validity, we selected different strategies for each step and implemented several strategies instead of focusing on a single one. Additionally, we tried to select simple strategies to reduce the impact of each strategy on the workflow.

External Validation As we could only conduct two case studies, and therefore create two instances of the xFogStar workflow, the degree of generalizability is low. Also the usage of the workflow resulted in the selection of the better fitting service provider based on the service consumers needs, the selection of the strategies for the different workflow parts might have influenced the results. We think that the workflow itself represents the needed steps to select the best fitting service provider and can therefore help to improve the setup of Fog Architecture. To be able to report on more generalizable results, the amount of instantiations of the xFogStar workflow has to be increased in different domains and with different strategies for each step.

Reliability The instantiations were created by students, only one single researcher analyzed the results of the case studies in relation to the workflow. We tried to mediate this threat by letting the students describe their findings in their theses while answering questions that might have come up during the conduction of the case study.

Chapter 7

Conclusion

"Good design is a renaissance attitude that combines technology, cognitive science, human need, and beauty to produce something that the world didn't know it was missing."

— PAOLA ANTONELLI

This chapter concludes the dissertation. In Section 7.1, we summarize the contributions, revisiting the concepts introduced in Chapter 3, Chapter 4, and Chapter 5. We address topics that require further investigation as well as opportunities to improve the presented xFog framework in Section 7.2.

7.1 Contributions

The main goal of this dissertation was to create a framework that establishes a formalization for Fog Computing, integrates support for mobile applications and dynamic reconfigurability of Fog Architectures. We used the design science methodology to investigate this goal using software engineering techniques followed by the creation of treatment designs and treatment validations. In the following, we revisit these phases of the design cycle.

Problem Investigation

In the introduction, we presented the problems that Fog Computing faces and its duality between a concept and a software architecture, as well as its missing support for dynamics and scalability. These problems resulted in three knowledge goals to address software architectures and to investigate the implications of the developed framework. Five technical research goals formed the technical objective for this dissertation. We provided summaries into the concepts related to Fog Computing and xFog, an e<u>x</u>tension for <u>**Fog**</u> Computing, to define corresponding terms.

We summarized the presented problems and technical research goals in use cases for the involved actors and formalized the use cases into functional and non-functional requirements which represent design problems. These have to be fulfilled by xFog to be considered an extension for Fog Computing that complies to software architectures and supports dynamic and scalable behavior of Fog Components and layers.

xFog Framework

To address the gap between Fog Computing as a concept and Fog Computing as a software architecture, we presented a historical overview of the evolution of software architecture and compared definitions from different authors. After identifying the similarities between the definitions, we introduced the xFog framework which is an extension to formalize Fog Computing and settle it as a software architecture. We achieved the first *Knowledge Goal* as well as the first three *Technical Research Goals* as described in Section 1.2 with xFog, xFogCore, and xFogPlus. *Technical Research Goal 4* could be achieved by creating xFogStar.

xFogCore defined the *Component Set* represented by the Fog Set and the *Communication Set* which relate to the components and connectors of a software architecture. To define the Fog Set, xFogCore introduced the *Fog Component Set*, *Fog Visibility*, *Fog Horizon*, and *Fog Reachability*. These concepts describe the components of a Fog Architecture based on mathematical definitions. We showed how the component sets can be constrained to specific services that are offered or consumed, or that are of interest for a Fog Component, which allows the identification of layers within Fog Architectures. We defined the *Communication Set* as a set of *Communication Components* which are defined by the involved Fog Components and the used communication channel. The sets were put into context by a meta model on MOF level M2 including the basic building blocks of software architectures which allowed the interpretation of the sets as graphs.

xFogPlus introduced support for the dynamic addition of Fog Components to the Fog Architecture at runtime by redefining the idea of Fog Components and by providing definitions for the three layers: *Edge Layer*, *Fog Layer*, and *Cloud Layer*. Second, new layers can be described and added to the Fog Architecture enabling scalability. As the scalability increased the complexity of the Fog Architecture, we established the concept of different *Views* on the Fog Architecture to set a focus on different layers depending on the stakeholder's current interest which is represented as the *Abstraction Level Pointer*.

xFogStar defined a workflow for the service provider selection in dynamically scalable Fog Architectures which are described by the concepts of xFogCore and xFogPlus. The workflow is used to select the best fitting service provider for the service consumer's needs.

These needs are represented as a vector of QoS parameters which we defined and categorized according to their dependencies. We investigated the different steps of the xFogStar workflow to outline arising problems.

Validation of xFog

We validated three aspects that use the foundations and formalization of xFog to investigate *Knowledge Goal 2* and *Knowledge Goal 3*: *Dynamic Fog Components*, *Scalable Fog Architectures*, and the *Service Provider Selection*. For each aspect, we used a multiple case study to gather quantitative data on the feasibility of xFog, and thus xFogCore, xFogPlus, and xFogStar. *Dynamic Fog Components* and *Scalable Fog Architectures* related to xFogCore and xFogPlus, while the *Service Provider Selection* addressed the xFogStar workflow.

Each validation compared the expected results represented by the models of the Fog Architectures for each case study with the results provided by xFog. The first multiple case study investigated three cases from different domains to support generalizable conclusions. It demonstrated the feasibility of xFog and in particular xFogPlus by examining the addition of components at runtime. The second validation included three cases and showed the feasibility of the scalable concepts of xFogPlus by adding new layers to existing Fog Architectures. The resulting Fog Architectures were used to highlight the applicability of the *View*-concept which addresses complexity depending on the stakeholder's current point of interest. Finally, the validation of the *Service Provider Selection* highlighted the feasibility of xFogStar with two instantiations of the workflow.

7.2 Future Work

Based on the development of xFog, and respectively xFogCore, xFogPlus, and xFogStar, we opened up further research opportunities that could be investigated.

7.2.1 Automation of xFog

xFog formalizes Fog Computing by providing equivalents to the component and connectors that define software architectures. Using set theory, different properties of the included relations, and the locality of Fog Computing, Fog Architectures can be described and analyzed. In its current state, xFog is applied as a manual process evaluating the relations that form the involved sets of xFogCore and the extensions provided by xFogPlus.

Based on the graph definitions introduced in Section 3.4, we imaging that the sets and relations could be evaluated at runtime in a semi-automated way. While evaluating which components conform to the Fog Component definition, introduced in Section 4.1.1, most likely remains a manual process, the evaluation of the Fog Visibility, and therefore the related sets up to the Fog Set, could be automated using algorithms to travers graphs such as breadth-first (BFS) or depth-first search (DFS).

To develop such algorithms, two challenges have to be investigated: First, Fog Architectures tend to use a wide variety of communication channels as described in Section 3.3. Accordingly, to identify all Fog Components that are integrated in a Fog Architecture, the used communication channels have be known at any time during runtime or the algorithms have to be implemented for all potential communication channels that are supported by any Fog Component. In particular in the area of IoT with its huge amount of different devices, communication channels, and protocols, this is an interesting topic for further research.

Second, based on the selected communication channels, the algorithms need to be able to identify circles within the graphs as every bidirectional communication channel can be seen as such. Circles can exist that use one communication channel to get to a Fog Component and another communication channel to get back, requiring the algorithms to be communication channel independent.

7.2.2 Service Migration

During the development of xFogPlus, we introduced definitions for the layers of Fog Architectures. These definitions are based on the provided services that are consumed and provided on each layer. Using this service-based layer definition allowed us to describe the process of adding new layers within a Fog Architecture, even describing the process of creating a new Fog Architecture from a client-server architecture. We addressed the corresponding implications of the new layers in relation to the described sets and concepts: Starting with the *Service Set*, we added the new services provided by the new layer. We updated the *Service Set Mapping* to reflect these changes, but we did not describe how services are moved from one layer to another layer. This problem is better known under the term "Process Migration" and well-described in literature. Nevertheless, it would be interesting to investigate the effects of the process migration workflows in the context of xFogPlus, to see if their definitions can be applied in Fog Architectures described by xFog.

7.2.3 Further research for xFogStar

Not only service consumers, but also service providers can have an interest in selecting the best fitting communication partner: Service providers might not have the capacities to provide the offered service to each potential service consumer which requests the offered services. In case of dynamic Fog Components, such a limited capacity is often the available energy. To address this issue, the xFogStar workflow could be extended to introduce a service negotiation between service consumers and service providers based on the xFog framework.

Second, based on the definitions of xFogStar, service providers guarantee for the provided parameters. This is an issue for parameters that are not solely depending on service providers. For instance, the bandwidth highly depends on the network setup, and therefore all hops in between the service consumer and service provider with the minimum bandwidth between those resulting in the overall bandwidth as described in Section 5.2.1. Accordingly, service providers often do not have an influence on these parameters.

If service providers know that they have bad QoS parameters, they could intentionally provide no or wrong values to get more requests from service consumers. In Section 5.2.2, we presented techniques to handle parameters that were not provided at all. This addresses the issue when service providers assume their values to be worse than the values from other service providers, and thus not sending theirs. One possible solution could be to establish a trusted instance in the Fog Architecture that stores the service consumers' ratings of the services. This idea is used in online markets, as for instance with the D-T scale [146].

The main drawback of this solution is that it is difficult for new service providers to establish themselves in the market and that before a rating can be created, some service consumers might have bad experiences with service providers. The second case, when service providers send wrong values is even worse, as the service consumers could loose trust in the xFogStar workflow and stop using it. Therefore, further research is required.

Finally, the xFogStar workflow requires additional time to establish a connection between a service consumer and service provider. This overhead in startup time poses another issue for xFogStar in particular in real-time applications that depend on fast discoveries and response times. While the response time can be specified to be low using QoS parameters, the discovery time is increased by using xFogStar. One possible solution is to limit the investigated QoS parameters, but further research is required.

Appendices

Appendix A

Licenses

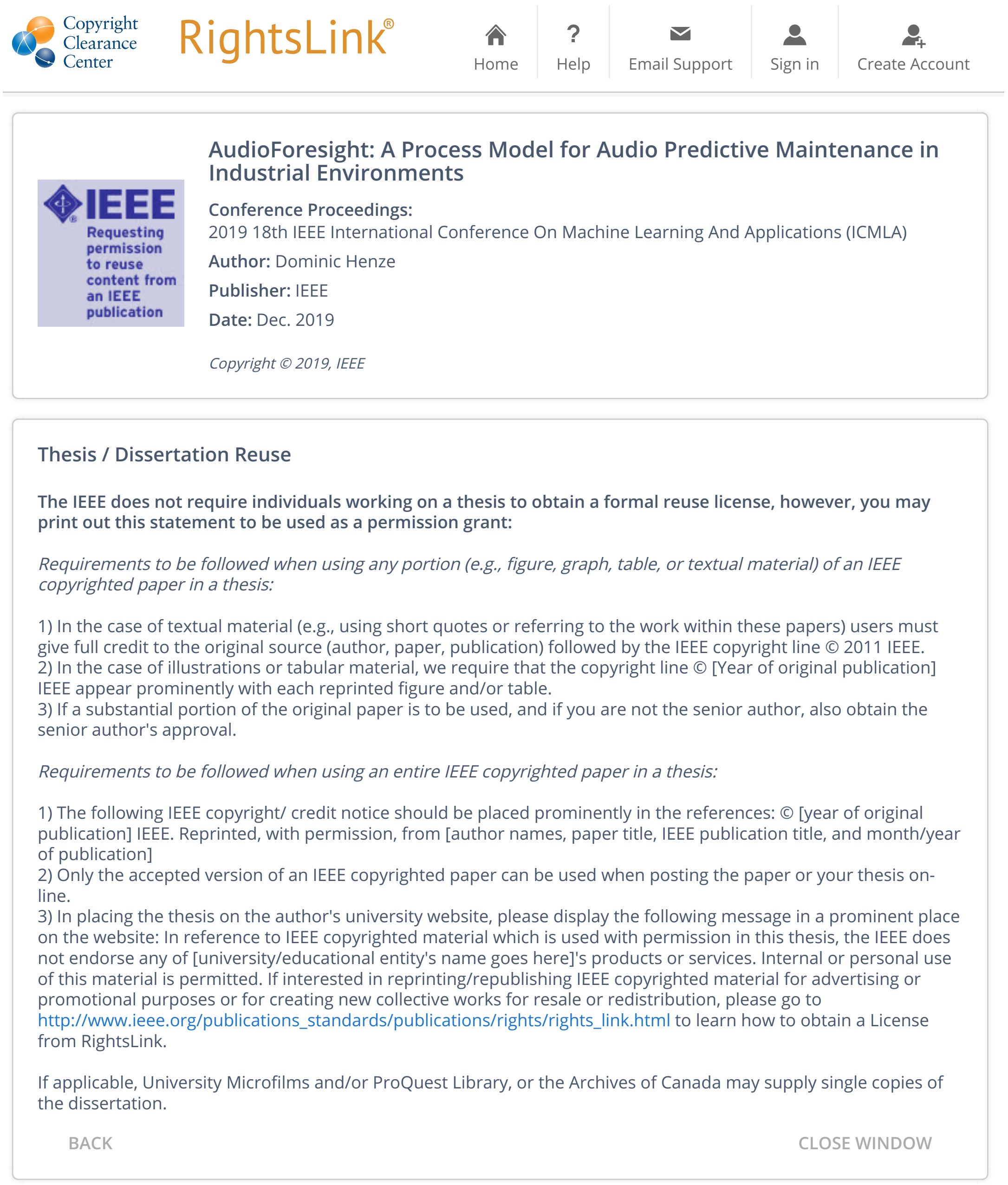
In the following, we list the licenses for the used material from previous publications.

A.1 ACM

Content of the paper [66] is used in this dissertation which is published under the ACM Author Rights agreement[35]. The following excerpt shows the relevant part for this dissertation:

[35]https://authors.acm.org/author-resources/author-rights

A.2 IEEE


Copyright Clearance Center
RightsLink®
Home
Help
Email Support
Sign in
Create Account
IEEE Requesting permission to reuse content from an IEEE publication

AudioForesight: A Process Model for Audio Predictive Maintenance in Industrial Environments

Conference Proceedings:
2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)

Author: Dominic Henze

Publisher: IEEE

Date: Dec. 2019

Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE WINDOW

Figure A.1: Permission grant for [65]

Appendix B

Additional Equations for Validation

This appendix provides additional equations for the results of the validation as shown in Chapter 6. For the validation of the dynamic Fog Components, we show the Communication Sets which were abbreviated for readability. For the scalable Fog Architectures validation we include the Fog Set and layer definitions, the written out form of the Communication Set, the adjustments to the Fog Set and layers after the new layer was added, and the unabbreviated Communication Set after the layer was added. For the validation of the service provider selection, we show the Fog Set, the Communication Set, the Service Set and Service Set Mapping, the layers, and the equations needed for the addition of a new component.

- *ARControl* is extended in Section B.1.
- *Lassie* is extended in Section B.2.
- *PdMFrame* is extended in Section B.3.
- *DisCoFog 1* is extended in Section B.4.
- *DisCoFog 2* is extended in Section B.5.
- *eHealth* is extended in Section B.6.
- *Fog.BOI* is extended in Section B.7.
- *Quasar* is extended in Section B.8.
- *FoQsIs* is extended in Section B.9.

B.1 ARControl

Equation 87 shows the written out form of the Communication Set for *ARControl* which was abbreviated for readability in Section 6.2.1.

Equation 87: ARControl: Communication Set

$$
\begin{aligned}
&CommunicationSet = \{\\
&\quad (OpenHABServer, Ethernet, Paul'sOffice),\\
&\quad (Paul'sOffice, Ethernet, OpenHABServer),\\
&\quad (OpenHABServer, Ethernet, Kitchen),\\
&\quad (Kitchen, Ethernet, OpenHABServer),\\
&\quad (OpenHABServer, Ethernet, ConferenceRoom),\\
&\quad (ConferenceRoom, Ethernet, OpenHABServer),\\
&\quad (Paul'sOffice, Ethernet, CeilingLight1),\\
&\quad (CeilingLight1, Ethernet, Paul'sOffice),\\
&\quad (Paul'sOffice, Ethernet, DeskLight1),\\
&\quad (DeskLight1, Ethernet, Paul'sOffice),\\
&\quad (Paul'sOffice, Ethernet, Occupancy1),\\
&\quad (Occupancy1, Ethernet, Paul'sOffice),\\
&\quad (Paul'sOffice, Ethernet, DeskHeater1),\\
&\quad (DeskHeater1, Ethernet, Paul'sOffice),\\
&\quad (Paul'sOffice, Ethernet, Temperature1),\\
&\quad (Temperature1, Ethernet, Paul'sOffice),\\
&\quad (Kitchen, Ethernet, CeilingLight2),\\
&\quad (CeilingLight2, Ethernet, Kitchen),\\
&\quad (Kitchen, Ethernet, Occupancy2),\\
&\quad (Occupancy2, Ethernet, Kitchen),\\
&\quad (Kitchen, Ethernet, Temperature2),\\
&\quad (Temperature2, Ethernet, Kitchen),\\
&\quad (ConferenceRoom, Ethernet, CeilingLight3),\\
&\quad (CeilingLight3, Ethernet, ConferenceRoom),\\
&\quad (ConferenceRoom, Ethernet, Window1),\\
&\quad (Window1, Ethernet, ConferenceRoom),\\
&\quad (ConferenceRoom, Ethernet, Occupancy3),\\
&\quad (Occupancy3, Ethernet, ConferenceRoom),\\
&\quad (ConferenceRoom, Ethernet, Window2),\\
&\quad (Window2, Ethernet, ConferenceRoom),\\
&\quad (ConferenceRoom, Ethernet, Temperature3),\\
&\quad (Temperature3, Ethernet, ConferenceRoom),\\
&\quad (ConferenceRoom, Ethernet, Window3),\\
&\quad (Window3, Ethernet, ConferenceRoom)\\
&\}
\end{aligned}
$$

B.2 Lassie

Equation 88 shows the written out form of the Communication Set for the *Lassie* case study.

Equation 88: Lassie: Communication Set

$$
\begin{aligned}
&\mathit{CommunicationSet} = \{\\
&\quad (\mathit{DRKServer}, \mathit{Ethernet}, \mathit{Patient1Home})\\
&\quad (\mathit{Patient1Home}, \mathit{Ethernet}, \mathit{DRKServer})\\
&\quad (\mathit{DRKServer}, \mathit{Ethernet}, \mathit{Patient2Home}),\\
&\quad (\mathit{Patient2Home}, \mathit{Ethernet}, \mathit{DRKServer}),\\
&\quad (\mathit{Patient1Home}, \mathit{WIFI}, \mathit{Door1}),\\
&\quad (\mathit{Door1}, \mathit{WIFI}, \mathit{Patient1Home}),\\
&\quad (\mathit{Patient1Home}, \mathit{WIFI}, \mathit{Motion1}),\\
&\quad (\mathit{Motion1}, \mathit{WIFI}, \mathit{Patient1Home}),\\
&\quad (\mathit{Patient1Home}, \mathit{WIFI}, \mathit{Occupancy1}),\\
&\quad (\mathit{Occupancy1}, \mathit{WIFI}, \mathit{Patient1Home}),\\
&\quad (\mathit{Patient1Home}, \mathit{WIFI}, \mathit{Heartrate1}),\\
&\quad (\mathit{Heartrate1}, \mathit{WIFI}, \mathit{Patient1Home}),\\
&\quad (\mathit{Patient1Home}, \mathit{WIFI}, \mathit{FallDetection1}),\\
&\quad (\mathit{FallDetection1}, \mathit{WIFI}, \mathit{Patient1Home}),\\
&\quad (\mathit{Patient2Home}, \mathit{WIFI}, \mathit{Door2}),\\
&\quad (\mathit{Door2}, \mathit{WIFI}, \mathit{Patient2Home}),\\
&\quad (\mathit{Patient2Home}, \mathit{WIFI}, \mathit{Motion2}),\\
&\quad (\mathit{Motion2}, \mathit{WIFI}, \mathit{Patient2Home}),\\
&\quad (\mathit{Patient2Home}, \mathit{WIFI}, \mathit{Occupancy2}),\\
&\quad (\mathit{Occupancy2}, \mathit{WIFI}, \mathit{Patient2Home}),\\
&\quad (\mathit{Patient2Home}, \mathit{WIFI}, \mathit{Heartrate2}),\\
&\quad (\mathit{Heartrate2}, \mathit{WIFI}, \mathit{Patient2Home}),\\
&\quad (\mathit{Patient2Home}, \mathit{WIFI}, \mathit{FallDetection2}),\\
&\quad (\mathit{FallDetection2}, \mathit{WIFI}, \mathit{Patient2Home})\\
&\}
\end{aligned}
$$

B.3 PdMFrame

The unabbreviated form of the Communication Set for *PdMFrame* is presented in Equation 89.

Equation 89: PdMFrame: Communication Set

$$
\begin{aligned}
&\mathit{CommunicationSet} = \{\\
&\quad (\mathit{DataAnalysisServer}, \mathit{Ethernet}, \mathit{AccessPoint1}),\\
&\quad (\mathit{AccessPoint1}, \mathit{Ethernet}, \mathit{DataAnalysisServer}),\\
&\quad (\mathit{DataAnalysisServer}, \mathit{Ethernet}, \mathit{AccessPoint2}),\\
&\quad (\mathit{AccessPoint2}, \mathit{Ethernet}, \mathit{DataAnalysisServer}),\\
&\quad (\mathit{AccessPoint1}, \mathit{Ethernet}, \mathit{DuraMax1}),\\
&\quad (\mathit{DuraMax1}, \mathit{Ethernet}, \mathit{AccessPoint1}),\\
&\quad (\mathit{AccessPoint1}, \mathit{Ethernet}, \mathit{Microphone1}),\\
&\quad (\mathit{Microphone1}, \mathit{Ethernet}, \mathit{AccessPoint1}),\\
&\quad (\mathit{AccessPoint1}, \mathit{Ethernet}, \mathit{Temperature1}),\\
&\quad (\mathit{Temperature1}, \mathit{Ethernet}, \mathit{AccessPoint1}),\\
&\quad (\mathit{AccessPoint1}, \mathit{Ethernet}, \mathit{Motion1}),\\
&\quad (\mathit{Motion1}, \mathit{Ethernet}, \mathit{AccessPoint1}),\\
&\quad (\mathit{AccessPoint2}, \mathit{Ethernet}, \mathit{DuraMax2}),\\
&\quad (\mathit{DuraMax2}, \mathit{Ethernet}, \mathit{AccessPoint2}),\\
&\quad (\mathit{AccessPoint2}, \mathit{Ethernet}, \mathit{Microphone2}),\\
&\quad (\mathit{Microphone2}, \mathit{Ethernet}, \mathit{AccessPoint2}),\\
&\quad (\mathit{AccessPoint2}, \mathit{Ethernet}, \mathit{Temperature2}),\\
&\quad (\mathit{Temperature2}, \mathit{Ethernet}, \mathit{AccessPoint2}),\\
&\quad (\mathit{AccessPoint2}, \mathit{Ethernet}, \mathit{Motion2}),\\
&\quad (\mathit{Motion2}, \mathit{Ethernet}, \mathit{AccessPoint2})\\
&\}
\end{aligned}
$$

B.4 DisCoFog

This section adds the equations to the *DisCoFog 1* case study results which were not the focus of the case study itself but are required as foundations for xFog.

- Equation 90 shows the Fog Set and the mapping of the Fog Components to their according layer.
- Equation 91 shows the unabbreviated Communication Set.
- Equation 92 shows the Fog Set and layer adjustments after a new layer was added to the Fog Architecture.
- Equation 93 shows the extended Communication Set after the addition of the new layer.

Equation 90: DisCoFog: Fog Set and Layers

$$\begin{aligned}FogSet = \{&DataVisualizationServer, City1, City2, Drone1, Drone2,\\ &Drone3, Drone4, Drone5, Drone6, Drone7, Drone8\}\end{aligned}$$

$$\begin{aligned}&CloudLayer = \{DataVisualizationServer\}\\ &FogLayer = \{City1, City2\}\\ &EdgeLayer = \{Drone1, Drone2, Drone3, Drone4,\\ &\qquad Drone5, Drone6, Drone7, Drone8\}\end{aligned}$$

Equation 91: DisCoFog: Communication Set

$$\begin{aligned}CommunicationSet = \{&(DataVisualizationServer, 3G/4G, City1),\\ &(City1, 3G/4G, DataVisualizationServer),\\ &(DataVisualizationServer, 3G/4G, City2),\\ &(City2, 3G/4G, DataVisualizationServer),\\ &(City1, WIFI, Drone1), (Drone1, WIFI, City1),\\ &(City1, WIFI, Drone2), (Drone2, WIFI, City1),\\ &(City1, WIFI, Drone3), (Drone3, WIFI, City1),\\ &(City1, WIFI, Drone4), (Drone4, WIFI, City1),\\ &(City1, WIFI, Drone5), (Drone5, WIFI, City1),\\ &(City1, WIFI, Drone6), (Drone6, WIFI, City1),\\ &(City2, WIFI, Drone7), (Drone7, WIFI, City2),\\ &(City2, WIFI, Drone8), (Drone8, WIFI, City2)\}\end{aligned}$$

Equation 92: DisCoFog: Fog Set and Layers after Layer Addition

$$\mathsf{FogSet}_{new} = \mathsf{FogSet}_{old} \cup \{\mathsf{City1Access1}, \mathsf{City1Access2}, \mathsf{City2Access1}\}$$

$$\begin{aligned}
&\mathsf{CloudLayer}_{new} = \mathsf{CloudLayer}_{old}\\
&\mathsf{EdgeLayer}_{new} = \mathsf{EdgeLayer}_{old}\\
&\mathsf{FogLayer2} = \mathsf{FogLayer}_{old}\\
&\mathsf{FogLayer1} = \{\mathsf{City1Access1}, \mathsf{City1Access2}, \mathsf{City2Access1}\}
\end{aligned}$$

Equation 93: DisCoFog: Communication Set after Layer Addition

$$\begin{aligned}
CommunicationSet = \{ & (DataVisualizationServer, 3G/4G, City1), \\
& (City1, 3G/4G, DataVisualizationServer), \\
& (DataVisualizationServer, 3G/4G, City2), \\
& (City2, 3G/4G, DataVisualizationServer), \\
& (City1, WIFI, City1Access1) \\
& (City1Access1, WIFI, City1) \\
& (City1, WIFI, City1Access2) \\
& (City1Access2, WIFI, City1) \\
& (City2, WIFI, City2Access1) \\
& (City2Access1, WIFI, City2) \\
& (City1Access1, WIFI, Drone1), \\
& (Drone1, WIFI, City1Access1), \\
& (City1Access1, WIFI, Drone2), \\
& (Drone2, WIFI, City1Access1), \\
& (City1Access1, WIFI, Drone3), \\
& (Drone3, WIFI, City1Access1), \\
& (City1Access2, WIFI, Drone4), \\
& (Drone4, WIFI, City1Access2), \\
& (City1Access2, WIFI, Drone5), \\
& (Drone5, WIFI, City1Access2), \\
& (City1Access2, WIFI, Drone6), \\
& (Drone6, WIFI, City1Access2), \\
& (City2Access1, WIFI, Drone7), \\
& (Drone7, WIFI, City2Access1), \\
& (City2Access1, WIFI, Drone8), \\
& (Drone8, WIFI, City2Access1)\}
\end{aligned}$$

B.5 DisCoFog 2

This section adds those equations to the *DisCoFog 2* case study results that are required for the foundations of xFog, but are not emphasized by the case study itself.

- Equation 94 shows the Fog Set and its layers.
- Equation 95 shows the Communication Set.
- Equation 96 shows the redefined Fog Set and its layers after the layer addition.
- Equation 97 shows the updated Communication Set after the layer addition.

Equation 94: DisCoFog 2: Fog Set and Layers

$$\begin{aligned} FogSet = \{&OpenHABServer, Paul'sOffice, Kitchen, ConferenceRoom,\\ &CeilingLight1, DeskLight1, Occupancy1, DeskHeater1,\\ &Temperature1, CeilingLight2, Occupancy2,\\ &Temperature2, CeilingLight3, Window1,\\ &Occupancy3, Window2, Temperature3,\\ &Window3\} \end{aligned}$$

$$\begin{aligned} CloudLayer &= \{OpenHABServer\}\\ FogLayer &= \{Paul'sOffice, Kitchen, ConferenceRoom\}\\ EdgeLayer &= \{CeilingLight1, DeskLight1, Occupancy1, DeskHeater1,\\ &\quad Temperature1, CeilingLight2, Occupancy2,\\ &\quad Temperature2, CeilingLight3, Window1,\\ &\quad Occupancy3, Window2,\\ &\quad Temperature3, Window3\} \end{aligned}$$

Equation 95: DisCoFog 2: Communication Set

$$
\begin{aligned}
&\mathit{CommunicationSet} = \{\\
&\quad(\mathit{OpenHABServer}, \mathit{Ethernet}, \mathit{Paul'sOffice}),\\
&\quad(\mathit{Paul'sOffice}, \mathit{Ethernet}, \mathit{OpenHABServer}),\\
&\quad(\mathit{OpenHABServer}, \mathit{Ethernet}, \mathit{Kitchen}),\\
&\quad(\mathit{Kitchen}, \mathit{Ethernet}, \mathit{OpenHABServer}),\\
&\quad(\mathit{OpenHABServer}, \mathit{Ethernet}, \mathit{ConferenceRoom}),\\
&\quad(\mathit{ConferenceRoom}, \mathit{Ethernet}, \mathit{OpenHABServer}),\\
&\quad(\mathit{Paul'sOffice}, \mathit{Ethernet}, \mathit{CeilingLight1}),\\
&\quad(\mathit{CeilingLight1}, \mathit{Ethernet}, \mathit{Paul'sOffice}),\\
&\quad(\mathit{Paul'sOffice}, \mathit{Ethernet}, \mathit{DeskLight1}),\\
&\quad(\mathit{DeskLight1}, \mathit{Ethernet}, \mathit{Paul'sOffice}),\\
&\quad(\mathit{Paul'sOffice}, \mathit{Ethernet}, \mathit{Occupancy1}),\\
&\quad(\mathit{Occupancy1}, \mathit{Ethernet}, \mathit{Paul'sOffice}),\\
&\quad(\mathit{Paul'sOffice}, \mathit{Ethernet}, \mathit{DeskHeater1}),\\
&\quad(\mathit{DeskHeater1}, \mathit{Ethernet}, \mathit{Paul'sOffice}),\\
&\quad(\mathit{Paul'sOffice}, \mathit{Ethernet}, \mathit{Temperature1}),\\
&\quad(\mathit{Temperature1}, \mathit{Ethernet}, \mathit{Paul'sOffice}),\\
&\quad(\mathit{Kitchen}, \mathit{Ethernet}, \mathit{CeilingLight2}),\\
&\quad(\mathit{CeilingLight2}, \mathit{Ethernet}, \mathit{Kitchen}),\\
&\quad(\mathit{Kitchen}, \mathit{Ethernet}, \mathit{Occupancy2}),\\
&\quad(\mathit{Occupancy2}, \mathit{Ethernet}, \mathit{Kitchen}),\\
&\quad(\mathit{Kitchen}, \mathit{Ethernet}, \mathit{Temperature2}),\\
&\quad(\mathit{Temperature2}, \mathit{Ethernet}, \mathit{Kitchen}),\\
&\quad(\mathit{ConferenceRoom}, \mathit{Ethernet}, \mathit{CeilingLight3}),\\
&\quad(\mathit{CeilingLight3}, \mathit{Ethernet}, \mathit{ConferenceRoom}),\\
&\quad(\mathit{ConferenceRoom}, \mathit{Ethernet}, \mathit{Window1}),\\
&\quad(\mathit{Window1}, \mathit{Ethernet}, \mathit{ConferenceRoom}),\\
&\quad(\mathit{ConferenceRoom}, \mathit{Ethernet}, \mathit{Occupancy3}),\\
&\quad(\mathit{Occupancy3}, \mathit{Ethernet}, \mathit{ConferenceRoom}),\\
&\quad(\mathit{ConferenceRoom}, \mathit{Ethernet}, \mathit{Window2}),\\
&\quad(\mathit{Window2}, \mathit{Ethernet}, \mathit{ConferenceRoom}),\\
&\quad(\mathit{ConferenceRoom}, \mathit{Ethernet}, \mathit{Temperature3}),\\
&\quad(\mathit{Temperature3}, \mathit{Ethernet}, \mathit{ConferenceRoom}),\\
&\quad(\mathit{ConferenceRoom}, \mathit{Ethernet}, \mathit{Window3}),\\
&\quad(\mathit{Window3}, \mathit{Ethernet}, \mathit{ConferenceRoom})\\
&\}
\end{aligned}
$$

Equation 96: DisCoFog 2: Fog Set and Layers after Layer Addition

$$\mathsf{FogSet}_{new} = \mathsf{FogSet}_{old} \cup \{\mathsf{Floor1}\}$$

$$\mathsf{CloudLayer}_{new} = \mathsf{CloudLayer}_{old}$$
$$\mathsf{EdgeLayer}_{new} = \mathsf{EdgeLayer}_{old}$$
$$\mathsf{FogLayer1} = \mathsf{FogLayer}_{old}$$
$$\mathsf{FogLayer2} = \{\mathsf{Floor1}\}$$

Equation 97: DisCoFog 2: Communication Set after Layer Addition

$$
\begin{aligned}
&CommunicationSet = \{\\
&\quad (OpenHABServer, LAN, Floor1),\\
&\quad (Floor1, LAN, OpenHABServer),\\
&\quad (Floor1, WIFI, Paul'sOffice),\\
&\quad (Paul'sOffice, WIFI, Floor1),\\
&\quad (Floor1, WIFI, Kitchen),\\
&\quad (Kitchen, WIFI, Floor1),\\
&\quad (Floor1, WIFI, ConferenceRoom),\\
&\quad (ConferenceRoom, WIFI, Floor1),\\
&\quad (Paul'sOffice, LAN, CeilingLight1),\\
&\quad (CeilingLight1, LAN, Paul'sOffice),\\
&\quad (Paul'sOffice, LAN, DeskLight1),\\
&\quad (DeskLight1, LAN, Paul'sOffice),\\
&\quad (Paul'sOffice, LAN, Occupancy1),\\
&\quad (Occupancy1, LAN, Paul'sOffice),\\
&\quad (Paul'sOffice, LAN, DeskHeater1),\\
&\quad (DeskHeater1, LAN, Paul'sOffice),\\
&\quad (Paul'sOffice, LAN, Temperature1),\\
&\quad (Temperature1, LAN, Paul'sOffice),\\
&\quad (Kitchen, LAN, CeilingLight2),\\
&\quad (CeilingLight2, LAN, Kitchen),\\
&\quad (Kitchen, LAN, Occupancy2),\\
&\quad (Occupancy2, LAN, Kitchen),\\
&\quad (Kitchen, LAN, Temperature2),\\
&\quad (Temperature2, LAN, Kitchen),\\
&\quad (ConferenceRoom, LAN, CeilingLight3),\\
&\quad (CeilingLight3, LAN, ConferenceRoom),\\
&\quad (ConferenceRoom, LAN, Window1),\\
&\quad (Window1, LAN, ConferenceRoom),\\
&\quad (ConferenceRoom, LAN, Occupancy3),\\
&\quad (Occupancy3, LAN, ConferenceRoom),\\
&\quad (ConferenceRoom, LAN, Window2),\\
&\quad (Window2, LAN, ConferenceRoom),\\
&\quad (ConferenceRoom, LAN, Temperature3),\\
&\quad (Temperature3, LAN, ConferenceRoom),\\
&\quad (ConferenceRoom, LAN, Window3),\\
&\quad (Window3, LAN, ConferenceRoom)\\
&\}
\end{aligned}
$$

B.6 eHealth

This section adds the fundamental equations to the *eHealth* case study results.

- Equation 98 shows the Fog Set and the according layers.
- Equation 99 shows the unabbreviated Communication Set.
- Equation 100 shows the updated Fog Set and its layer after the layer addition.
- Equation 101 shows the updated Communication Set after the layer addition.

Equation 98: eHealth: Fog Set and Layers

$$\begin{aligned} FogSet = \{&HospitalServer, Floor1, Floor2, Room1,\\ &Room2, Room3, Microscope, RoboticArm,\\ &RadiologyDevice, DiagnosticDevice\} \end{aligned}$$

$$CloudLayer = \{HospitalServer\}$$
$$FogLayer_2 = \{Floor1, Floor2\}$$
$$FogLayer_1 = \{Room1, Room2, Room3\}$$
$$\begin{aligned} EdgeLayer = \{&Microscope, RoboticArm, RadiologyDevice,\\ &DiagnosticDevice\} \end{aligned}$$

Equation 99: eHealth: Communication Set

$$
\begin{aligned}
&\mathrm{CommunicationSet} = \{\\
&\quad (\mathrm{HospitalServer}, \mathrm{Ethernet}, \mathrm{Floor1}),\\
&\quad (\mathrm{Floor1}, \mathrm{Ethernet}, \mathrm{HospitalServer}),\\
&\quad (\mathrm{HospitalServer}, \mathrm{Ethernet}, \mathrm{Floor2}),\\
&\quad (\mathrm{Floor2}, \mathrm{Ethernet}, \mathrm{HospitalServer}),\\
&\quad (\mathrm{Floor1}, \mathrm{Ethernet}, \mathrm{Room1}),\\
&\quad (\mathrm{Room1}, \mathrm{Ethernet}, \mathrm{Floor1}),\\
&\quad (\mathrm{Floor1}, \mathrm{Ethernet}, \mathrm{Room2}),\\
&\quad (\mathrm{Room2}, \mathrm{Ethernet}, \mathrm{Floor1}),\\
&\quad (\mathrm{Floor2}, \mathrm{Ethernet}, \mathrm{Room3}),\\
&\quad (\mathrm{Room3}, \mathrm{Ethernet}, \mathrm{Floor2}),\\
&\quad (\mathrm{Room1}, \mathrm{WIFI}, \mathrm{Microscope}),\\
&\quad (\mathrm{Microscope}, \mathrm{WIFI}, \mathrm{Room1}),\\
&\quad (\mathrm{Room2}, \mathrm{WIFI}, \mathrm{RoboticArm}),\\
&\quad (\mathrm{RoboticArm}, \mathrm{WIFI}, \mathrm{Room2}),\\
&\quad (\mathrm{Room2}, \mathrm{WIFI}, \mathrm{RadiologyDevice}),\\
&\quad (\mathrm{RadiologyDevice}, \mathrm{WIFI}, \mathrm{Room2}),\\
&\quad (\mathrm{Room3}, \mathrm{WIFI}, \mathrm{DiagnosticDevice}),\\
&\quad (\mathrm{DiagnosticDevice}, \mathrm{WIFI}, \mathrm{Room3})\\
&\}
\end{aligned}
$$

Equation 100: eHealth: Fog Set and Layers after Layer Addition

$$\mathrm{FogSet}_{new} = \mathrm{FogSet}_{old} \cup \{\mathrm{Department1}\}$$

$$
\begin{aligned}
&\mathrm{CloudLayer}_{new} = \mathrm{CloudLayer}_{old}\\
&\mathrm{EdgeLayer}_{new} = \mathrm{EdgeLayer}_{old}\\
&\mathrm{FogLayer1} = \mathrm{FogLayer1}_{old}\\
&\mathrm{FogLayer2} = \mathrm{FogLayer2}_{old}\\
&\mathrm{FogLayer3} = \{\mathrm{Department1}\}
\end{aligned}
$$

Equation 101: eHealth: Communication Set after Layer Addition

$$
\begin{aligned}
&\mathit{CommunicationSet} = \{\\
&\quad (\mathit{HospitalServer}, \mathit{LAN}, \mathit{Department1}),\\
&\quad (\mathit{Department1}, \mathit{LAN}, \mathit{HospitalServer}),\\
&\quad (\mathit{Department1}, \mathit{LAN}, \mathit{Floor1}),\\
&\quad (\mathit{Floor1}, \mathit{LAN}, \mathit{Department1}),\\
&\quad (\mathit{Department1}, \mathit{LAN}, \mathit{Floor2}),\\
&\quad (\mathit{Floor2}, \mathit{LAN}, \mathit{Department1}),\\
&\quad (\mathit{Floor1}, \mathit{LAN}, \mathit{Room1}),\\
&\quad (\mathit{Room1}, \mathit{LAN}, \mathit{Floor1}),\\
&\quad (\mathit{Floor1}, \mathit{LAN}, \mathit{Room2}),\\
&\quad (\mathit{Room2}, \mathit{LAN}, \mathit{Floor1}),\\
&\quad (\mathit{Floor2}, \mathit{LAN}, \mathit{Room3}),\\
&\quad (\mathit{Room3}, \mathit{LAN}, \mathit{Floor2}),\\
&\quad (\mathit{Room1}, \mathit{WIFI}, \mathit{Microscope}),\\
&\quad (\mathit{Microscope}, \mathit{WIFI}, \mathit{Room1}),\\
&\quad (\mathit{Room2}, \mathit{WIFI}, \mathit{RoboticArm}),\\
&\quad (\mathit{RoboticArm}, \mathit{WIFI}, \mathit{Room2}),\\
&\quad (\mathit{Room2}, \mathit{WIFI}, \mathit{RadiologyDevice}),\\
&\quad (\mathit{RadiologyDevice}, \mathit{WIFI}, \mathit{Room2}),\\
&\quad (\mathit{Room3}, \mathit{WIFI}, \mathit{DiagnosticDevice}),\\
&\quad (\mathit{DiagnosticDevice}, \mathit{WIFI}, \mathit{Room3})\\
&\}
\end{aligned}
$$

B.7 Fog.BOI

This section adds the equations for the foundations of xFog to the *Fog.BOI* case study results.

- Equation 102 shows the Fog Set and the according layers.
- Equation 103 shows the unabbreviated Communication Set.
- Equation 104 shows the updated Fog Set and its layer after the layer addition.
- Equation 105 shows the updated Communication Set after the layer addition.

Equation 102: Fog.BOI: Fog Set and Layers

$$\begin{aligned} FogSet = \{ & MunicipalityServer, TrafficLight1, TrafficLight2, \\ & Vehicle1, Vehicle2, Vehicle3, Vehicle4, Vehicle5, Vehicle6\} \end{aligned}$$

$$CloudLayer = \{MunicipalityServer\}$$
$$FogLayer = \{TrafficLight1, TrafficLight2\}$$
$$EdgeLayer = \{Vehicle1, Vehicle2, Vehicle3, Vehicle4, Vehicle5, Vehicle6\}$$

Equation 103: Fog.BOI: Communication Set

$$\begin{aligned} & CommunicationSet = \{ \\ & \quad (MunicipalityServer, Ethernet, TrafficLight1), \\ & \quad (TrafficLight1, Ethernet, MunicipalityServer), \\ & \quad (MunicipalityServer, Ethernet, TrafficLight2), \\ & \quad (TrafficLight2, Ethernet, MunicipalityServer), \\ & \quad (TrafficLight1, WIFI, Vehicle1), \\ & \quad (Vehicle1, WIFI, TrafficLight1), \\ & \quad (TrafficLight1, WIFI, Vehicle2), \\ & \quad (Vehicle2, WIFI, TrafficLight1), \\ & \quad (TrafficLight1, WIFI, Vehicle3), \\ & \quad (Vehicle3, WIFI, TrafficLight1), \\ & \quad (TrafficLight2, WIFI, Vehicle4), \\ & \quad (Vehicle4, WIFI, TrafficLight2), \\ & \quad (TrafficLight2, WIFI, Vehicle5), \\ & \quad (Vehicle5, WIFI, TrafficLight2), \\ & \quad (TrafficLight2, WIFI, Vehicle6), \\ & \quad (Vehicle6, WIFI, TrafficLight2) \\ & \} \end{aligned}$$

Equation 104: Fog.BOI: Fog Set and Layers after Layer Addition

$$\mathsf{FogSet}_{new} = \mathsf{FogSet}_{old} \cup \{\mathsf{CityDistrict}\}$$

$$\begin{aligned}
&\mathsf{CloudLayer}_{new} = \mathsf{CloudLayer}_{old}\\
&\mathsf{EdgeLayer}_{new} = \mathsf{EdgeLayer}_{old}\\
&\mathsf{FogLayer1} = \mathsf{FogLayer}_{old}\\
&\mathsf{FogLayer2} = \{\mathsf{CityDistrict}\}
\end{aligned}$$

Equation 105: Fog.BOI: Communication Set after Layer Addition

$$\begin{aligned}
\mathsf{CommunicationSet} = \{&\\
&(\mathsf{MunicipalityServer}, \mathsf{Ethernet}, \mathsf{CityDistrict}),\\
&(\mathsf{CityDistrict}, \mathsf{Ethernet}, \mathsf{MunicipalityServer}),\\
&(\mathsf{CityDistrict}, \mathsf{Ethernet}, \mathsf{TrafficLight2}),\\
&(\mathsf{TrafficLight2}, \mathsf{Ethernet}, \mathsf{CityDistrict}),\\
&(\mathsf{CityDistrict}, \mathsf{Ethernet}, \mathsf{TrafficLight2}),\\
&(\mathsf{TrafficLight2}, \mathsf{Ethernet}, \mathsf{CityDistrict}),\\
&(\mathsf{TrafficLight1}, \mathsf{WIFI}, \mathsf{Vehicle1}),\\
&(\mathsf{Vehicle1}, \mathsf{WIFI}, \mathsf{TrafficLight1}),\\
&(\mathsf{TrafficLight1}, \mathsf{WIFI}, \mathsf{Vehicle2}),\\
&(\mathsf{Vehicle2}, \mathsf{WIFI}, \mathsf{TrafficLight1}),\\
&(\mathsf{TrafficLight1}, \mathsf{WIFI}, \mathsf{Vehicle3}),\\
&(\mathsf{Vehicle3}, \mathsf{WIFI}, \mathsf{TrafficLight1}),\\
&(\mathsf{TrafficLight2}, \mathsf{WIFI}, \mathsf{Vehicle4}),\\
&(\mathsf{Vehicle4}, \mathsf{WIFI}, \mathsf{TrafficLight2}),\\
&(\mathsf{TrafficLight2}, \mathsf{WIFI}, \mathsf{Vehicle5}),\\
&(\mathsf{Vehicle5}, \mathsf{WIFI}, \mathsf{TrafficLight2}),\\
&(\mathsf{TrafficLight2}, \mathsf{WIFI}, \mathsf{Vehicle6}),\\
&(\mathsf{Vehicle6}, \mathsf{WIFI}, \mathsf{TrafficLight2})\\
\}&
\end{aligned}$$

B.8 Quasar

This section adds the equations for the foundations of xFog to the *Quasar* case study results. Additionally, the equations and the table required for the addition of a new Fog Component to a Fog Architecture are given in the same way as for the dynamic Fog Components validation.

- Equation 106 shows the Fog Set and the according layers.
- Equation 107 shows the unabbreviated Communication Set.
- Equation 108 shows the involved services as well as their mapping to the layers.
- Table B.1 checks the properties required for a component to comply with the Fog Component definition.
- Equation 109 shows the changes to the respective sets after a new Fog Component is added to the Fog Architecture.

Equation 106: Quasar: Fog Set

$$\begin{aligned} FogSet = \{&GameServer, GameStreamingService1, GameStreamingService2,\\ &GameClient1, GameClient2, GameClient3, GameClient4\} \end{aligned}$$

$$\begin{aligned} &CloudLayer = \{GameServer\}\\ &FogLayer = \{GameStreamingService1, GameStreamingService2\}\\ &EdgeLayer = \{GameClient1, GameClient2, GameClient3, GameClient4\} \end{aligned}$$

Equation 107: Quasar: Communication Set

$$\begin{aligned} CommunicationSet = \{&(GameServer, Ethernet, GameStreamingService1),\\ &(GameStreamingService1, Ethernet, GameServer),\\ &(GameServer, Ethernet, GameStreamingService2),\\ &(GameStreamingService2, Ethernet, GameServer),\\ &(GameStreamingService1, Ethernet, GameClient1),\\ &(GameClient1, Ethernet, GameStreamingService1),\\ &(GameStreamingService1, Ethernet, GameClient2),\\ &(GameClient2, Ethernet, GameStreamingService1),\\ &(GameStreamingService2, Ethernet, GameClient3),\\ &(GameClient3, Ethernet, GameStreamingService2),\\ &(GameStreamingService2, Ethernet, GameClient4),\\ &(GameClient4, Ethernet, GameStreamingService2)\} \end{aligned}$$

Table B.1: Hard and soft requirements of Fog Components according to Section 4.1 for the dynamic Fog Component: `Game Client 5`. Hard requirements are highlighted in *red*, soft requirements in *orange*.

		Game Client 5
1.	Interconnectivity	✓
2.	Information Sharing	✓
3.	Uniquely Addressable	✓
4.	Computational Capabilities	✓
5.	Wireless communication	✓
6.	Locality	✓
7.	General-purpose Computational Capabilities	✓
8.	Offers Capabilities as Service	X

Equation 108: Quasar: Service Set and Service Set Mapping

$$\begin{aligned}\mathsf{ServiceSet} = \{&\mathsf{getGlobalGameInstance}, \mathsf{setGlobalGameState},\\ &\mathsf{getGameInstance}, \mathsf{setGameState}\}\\ \mathsf{ServiceSetMapping} = \{&(\{\}, \mathsf{CloudLayer},\\ &\{\mathsf{getGlobalGameInstance}, \mathsf{setGlobalGameState}\}),\\ &(\{\mathsf{getGlobalGameInstance}, \mathsf{setGlobalGameState}\},\\ &\mathsf{FogLayer}, \{\mathsf{getGameInstance}, \mathsf{setGameState}\}),\\ &(\{\mathsf{getGameInstance}, \mathsf{setGameState}\}, \mathsf{EdgeLayer},\\ &\{\})\}\end{aligned}$$

Equation 109: Quasar: New Fog Component

$$\begin{aligned}\mathsf{FogSet}_{new} &= \mathsf{FogSet}_{old} \cup \{\mathsf{GameClient5}\}\\ \mathsf{CloudLayer}_{new} &= \mathsf{CloudLayer}_{old}\\ \mathsf{FogLayer}_{new} &= \mathsf{FogLayer}_{old}\\ \mathsf{EdgeLayer}_{new} &= \mathsf{EdgeLayer}_{old} \cup \{\mathsf{GameClient5}\}\\ \mathsf{CommunicationSet}_{new} &= \mathsf{CommunicationSet}_{old} \cup\\ &\{(\mathsf{GameStreamingService2}, \mathsf{WIFI}, \mathsf{GameClient5}),\\ &(\mathsf{GameClient5}, \mathsf{WIFI}, \mathsf{GameStreamingService2})\}\end{aligned}$$

B.9 FoQsIs

This section adds the equations for the foundations of xFog to the *FoQsIs* case study results. Additionally, the equations and the table required for the addition of a new Fog Component to a Fog Architecture are given in the same way as for the dynamic Fog Components validation.

- Equation 110 shows the Fog Set and the according layers.
- Equation 111 shows the unabbreviated Communication Set.
- Equation 112 shows the involved services as well as their mapping to the layers.
- Table B.2 checks the properties required for a component to comply with the Fog Component definition.
- Equation 113 shows the changes to the respective sets after a new Fog Component is added to the Fog Architecture.

Equation 110: FoQsIs: Fog Set

$$
\begin{aligned}
FogSet = \{&StatsServer, IntegrationService1, IntegrationService2,\\
&IntegrationService3, IntegrationService4, IntegrationService5,\\
&FoQsIsClient1, FoQsIsClient2\}
\end{aligned}
$$

$$
\begin{aligned}
&CloudLayer = \{StatsServer\}\\
&FogLayer = \{IntegrationService1, IntegrationService2, IntegrationService3,\\
&\qquad IntegrationService4, IntegrationService5\}\\
&EdgeLayer = \{FoQsIsClient1, FoQsIsClient2\}
\end{aligned}
$$

Equation 111: FoQsIs: Communication Set

$$\begin{aligned}
\text{CommunicationSet} = \{&(\text{StatsServer}, \text{Ethernet}, \text{IntegrationService1}),\\
&(\text{IntegrationService1}, \text{Ethernet}, \text{StatsServer}),\\
&(\text{StatsServer}, \text{Ethernet}, \text{IntegrationService2}),\\
&(\text{IntegrationService2}, \text{Ethernet}, \text{StatsServer}),\\
&(\text{StatsServer}, \text{Ethernet}, \text{IntegrationService3}),\\
&(\text{IntegrationService3}, \text{Ethernet}, \text{StatsServer}),\\
&(\text{StatsServer}, \text{Ethernet}, \text{IntegrationService4}),\\
&(\text{IntegrationService4}, \text{Ethernet}, \text{StatsServer}),\\
&(\text{StatsServer}, \text{Ethernet}, \text{IntegrationService5}),\\
&(\text{IntegrationService5}, \text{Ethernet}, \text{StatsServer}),\\
&(\text{StatsServer}, \text{WIFI}, \text{FoQsIsClient1}),\\
&(\text{FoQsIsClient1}, \text{WIFI}, \text{StatsServer}),\\
&(\text{StatsServer}, \text{WIFI}, \text{FoQsIsClient2}),\\
&(\text{FoQsIsClient2}, \text{WIFI}, \text{StatsServer})\}
\end{aligned}$$

Equation 112: FoQsIs: Service Set and Service Set Mapping

$$\begin{aligned}
\text{ServiceSet} = \{&\text{getIntegrationServiceStats}, \text{setIntegrationServiceStats},\\
&\text{performBuild}\}\\
\text{ServiceSetMapping} = \{(&\{\}, \text{CloudLayer},\\
&\{\text{getIntegrationServiceStats},\\
&\text{setIntegrationServiceStats}\}),\\
&\{\text{getIntegrationServiceStats},\\
&\text{setIntegrationServiceStats}\}),\\
&\text{FogLayer}, \{\text{performBuild}\}),\\
&(\{\text{performBuild}\}, \text{EdgeLayer}, \{\})\}
\end{aligned}$$

Table B.2: Hard and soft requirements of Fog Components according to Section 4.1 for the dynamic Fog Component: `FoQsIs Client 3`. Hard requirements are highlighted in *red*, soft requirements in *orange*.

		FoQsIs Client 3
1.	Interconnectivity	✓
2.	Information Sharing	✓
3.	Uniquely Addressable	✓
4.	Computational Capabilities	✓
5.	Wireless communication	✓
6.	Locality	✓
7.	General-purpose Computational Capabilities	✓
8.	Offers Capabilities as Service	X

Equation 113: FoQsIs: New Fog Component

$$\begin{aligned}
&\mathsf{FogSet}_{new} = \mathsf{FogSet}_{old} \cup \{\mathsf{FoQsIsClient3}\}\\
&\mathsf{CloudLayer}_{new} = \mathsf{CloudLayer}_{old}\\
&\mathsf{FogLayer}_{new} = \mathsf{FogLayer}_{old}\\
&\mathsf{EdgeLayer}_{new} = \mathsf{EdgeLayer}_{old} \cup \{\mathsf{FoQsIsClient3}\}\\
&\mathsf{CommunicationSet}_{new} = \mathsf{CommunicationSet}_{old} \cup\\
&\qquad \{(\mathsf{IntegrationService4}, \mathsf{WIFI}, \mathsf{FoQsIsClient3}),\\
&\qquad (\mathsf{FoQsIsClient3}, \mathsf{WIFI}, \mathsf{IntegrationService4})\}
\end{aligned}$$

List of Figures

List of Tables

Bibliography

[1] Shashank Agrawal and Melissa Chase. Fame: fast attribute-based message encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 665–682. ACM, 2017.

[2] Sebastian Aigner. Fog.boi - a scalable broker operation and interaction system for fog computing, bachelor's thesis, technical university of munich. Bachelor's thesis, Technical University of Munich, Munich, 2019.

[3] Mohammad Abdullah Al Faruque and Korosh Vatanparvar. Energy management-as-a-service over fog computing platform. *IEEE internet of things journal*, 3(2):161–169, 2015.

[4] Eyhab Al-Masri and Qusay Mahmoud. Qos-based discovery and ranking of web services. In *2007 16th international conference on computer communications and networks*, pages 529–534. IEEE, 2007.

[5] Robert Allen. *A Formal Approach to Software Architecture.* PhD thesis, Carnegie Mellon University, 1997.

[6] Stefano Alletto, Rita Cucchiara, Giuseppe Del Fiore, Luca Mainetti, Vincenzo Mighali, Luigi Patrono, and Giuseppe Serra. An indoor location-aware system for an iot-based smart museum. *IEEE Internet of Things Journal*, 3:244–253, 2015.

[7] Kofi Annan. The role of the united nations in the 21st century. *Report presented by the Secretary-General on*, 4, 2000.

[8] Sally A Applin and Michael D Fischer. Thing theory: Connecting humans to location-aware smart environments. *LAMDa'13*, page 29, 2013.

[9] Michael Armbrust, Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53:50–58, 2010.

[10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54:2787–2805, 2010.

[11] Arshdeep Bahga and Vijay Madisetti. Analyzing massive machine maintenance data in a computing cloud. *IEEE Transactions on Parallel and Distributed Systems Conference*, 23:1831–1843, 2012.

[12] Béla Bajnok. *An Invitation to Abstract Mathematics*, pages 229–239. Springer New York, 2013.

[13] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[14] Izak Benbasat, David K Goldstein, and Melissa Mead. The case research strategy in studies of information systems. *MIS quarterly*, pages 369–386, 1987.

[15] Glenn Bergland. A guided tour of program design methodologies. *Computer*, 14(10):13–37, 1981.

[16] Dean Bidgood Jr, Steven Horii, Fred Prior, and Donald Van Syckle. Understanding and using dicom, the data interchange standard for biomedical imaging. *Journal of the American Medical Informatics Association*, 4:199–212, 1997.

[17] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for internet of things and analytics. In *Big data and internet of things: A roadmap for smart environments*, pages 169–186. Springer, 2014.

[18] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[19] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. On the integration of cloud computing and internet of things. In *2014 International Conference on Future Internet of Things and Cloud*, pages 23–30. IEEE, 2014.

[20] Paul Brody and Veena Pureswaran. Device democracy: Saving the future of the internet of things. Technical report, IBM Institute, 2014.

[21] Bernd Bruegge and Allen Dutoit. *Object Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall, 3rd edition, 2010.

[22] Bernd Bruegge, Stephan Krusche, and Lukas Alperowitz. Software engineering project courses with industrial clients. *ACM Transactions on Computing Education (TOCE)*, 15:1–31, 2015.

[23] Juan Cadavid, Benoît Combemale, and Benoit Baudry. Ten years of metaobject facility: an analysis of metamodeling practices. *Research Report*, 2012.

[24] Herrn Cantor. Ueber eine eigenschaft des inbegriffes aller reellen algebraischen zahlen. In *Über unendliche, lineare Punktmannigfaltigkeiten*, pages 19–24. Springer, 1984.

[25] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. Distributed qos-aware scheduling in storm. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, pages 344–347, 2015.

[26] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. On qos-aware scheduling of data stream applications over fog computing infrastructures. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 271–276. IEEE, 2015.

[27] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *Journal of web semantics*, 1:281–308, 2004.

[28] Marc Castel and Corbin Church. System and method for providing consumer side maintenance. *United States Patent Application Publication*, 2014.

[29] Dazhi Chen and Pramod Varshney. Qos support in wireless sensor networks: A survey. In *International conference on wireless networks*, volume 233, pages 1–7, 2004.

[30] Stuart Cheshire and Marc Krochmal. Dns-based service discovery. Technical report, RFC Editor, 2013.

[31] Lawrence Chung, Brian Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.

[32] Cisco. Cisco global cloud index: Forecast and methodology, 2016–2021. White paper, Cisco, 2016.

[33] Giorgio Coricelli, Dietmar Fehr, and Gerlinde Fellner. Partner selection in public goods experiments. *Journal of Conflict Resolution*, 48:356–378, 2004.

[34] Stephen Crawley, S Davies, Jadwiga Indulska, Simon McBride, and Kerry Raymond. Meta-meta is better-better! In *DAIS*. Citeseer, 1997.

[35] John W Creswell and Cheryl N Poth. *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications, 2016.

[36] Nigel Cross. Designerly ways of knowing: Design discipline versus design science. *Design issues*, 17(3):49–55, 2001.

[37] Rene Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected areas in Communications*, 13:1048–1056, 1995.

[38] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh, and Rajkumar Buyya. Fog computing: Principles, architectures, and applications. In *Internet of things*, pages 61–75. Elsevier, 2016.

[39] Ruilong Deng, Rongxing Lu, Chengzhe Lai, Tom Luan, and Hao Liang. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. *IEEE internet of things journal*, 3(6):1171–1181, 2016.

[40] Edsger Dijkstra. The structure of the "the" multiprogramming system. In *The origin of concurrent programming*, pages 139–152. Springer, 1968.

[41] Edsger Dijkstra. The humble programmer. *Commun. ACM*, 15(10):859–866, 1972.

[42] Philipp Diller. A dynamically scalable fog architecture in a hospital setting, master's thesis, technical university of munich. Master's thesis, Technical University of Munich, Munich, 2020.

[43] Hoang Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[44] Koustabh Dolui and Soumya Kanti Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[45] Philipp Eichstetter. Qos based service negotiation for dynamically scalable fog architectures, bachelor's thesis, technical university of munich. Bachelor's thesis, Technical University of Munich, Munich, 2020.

[46] Ana Juan Ferrer, Joan Manuel Marquès, and Josep Jorba. Towards the Decentralised Cloud: Survey on Approaches and Challenges for Mobile, Ad hoc, and Edge Computing. *ACM Computing Surveys (CSUR)*, 51:111, 2019.

[47] Elgar Fleisch and Friedemann Mattern. *Das Internet der Dinge*. Springer-Verlag GmbH, 2005.

[48] Elgar Fleisch, Markus Weinberger, and Felix Wortmann. Geschäftsmodelle im internet der dinge. In *Industrie 4.0*, pages 1–16. Springer, 2017.

[49] Martin Fowler and Matthew Foemmel. Continuous integration, 2006.

[50] Svend Frølund and Jari Koistinen. Quality-of-service specification in distributed object systems. *Distributed Systems Engineering*, 5:179, 1998.

[51] Cristina Gacek, Ahmed Abd-Allah, Bradford Clark, and Barry Boehm. On the definition of software system architecture. In *Proceedings of the First International Workshop on Architectures for Software Systems*, pages 85–94. Seattle, Wa, 1995.

[52] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, et al. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 45(5):37–42, 2015.

[53] David Garlan, Robert Allen, and John Ockerbloom. Exploiting style in architectural design environments. *ACM SIGSOFT Software Engineering Notes*, 19(5):175–188, 1994.

[54] David Garlan and Mary Shaw. An introduction to software architecture. In *Advances in software engineering and knowledge engineering*, pages 1–39. World Scientific, 1993.

[55] Leonidas Georgiadis, Roch Guérin, Vinod Peris, and Kumar Sivarajan. Efficient network qos provisioning based on per node traffic shaping. *IEEE/ACM transactions on networking*, 4:482–501, 1996.

[56] Klaidi Gorishti. An audio-based approach for industrial equipment predictive maintenance, master's thesis, technical university of munich. Master's thesis, Technical University of Munich, Munich, 2018.

[57] Robert Grady. *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall, Englewood Cliffs, 1992.

[58] Object Management Group. Meta object facility. Technical report, Object Management Group, 2016.

[59] Sandra Grujovic. Qos based service discovery for dynamically scalable fog architectures, master's thesis, technical university of munich. Master's thesis, Technical University of Munich, Munich, 2019.

[60] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29:1645–1660, 2013.

[61] Arnt Gulbrandsen, Paul Vixie, and Levon Esibov. A dns rr for specifying the location of services (dns srv). Technical report, RFC Editor, 2000.

[62] Johanna Gustafsson. Single case studies vs. multiple case studies: A comparative study, 2017.

[63] Volker Hartkopf, Vivian Loftness, Ardeshir Mahdavi, Stephen Lee, and Jayakrishna Shankavaram. An integrated approach to design and engineering of intelligent buildings—the intelligent workplace at carnegie mellon university. *Automation in Construction*, 6:401–415, 1997.

[64] Mohammed Hassan, Mengbai Xiao, Qi Wei, and Songqing Chen. Help your mobile applications with fog computing. In *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking-Workshops (SECON Workshops)*, pages 1–6. IEEE, 2015.

[65] Dominic Henze, Klaidi Gorishti, Bernd Bruegge, and Jan-Philipp Simen. Audioforesight: A process model for audio predictive maintenance in industrial environments. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 352–357. IEEE, 2019.

[66] Dominic Henze, Paul Schmiedmayer, and Bernd Bruegge. Fog horizons–a theoretical concept to enable dynamic fog architectures. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 41–50, 2019.

[67] Alan Hevner, Salvatore March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.

[68] Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied software architecture*. Addison-Wesley Professional, 2000.

[69] Roger Horn. The hadamard product. In *Proc. Symp. Appl. Math*, volume 40, pages 87–169, 1990.

[70] Weigang Hou, Zhaolong Ning, and Lei Guo. Green survivable collaborative edge computing in smart cities. *IEEE Transactions on Industrial Informatics*, 14:1594–1605, 2018.

[71] Krittin Intharawijitr, Katsuyoshi Iida, and Hiroyuki Koga. Analysis of fog model considering computing and communication latency in 5g cellular networks. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–4, 2016.

[72] ISO ISO. Iec/ieee systems and software engineering: Architecture description. *ISO/IEC/IEEE 42010: 2011 (E)(Revision of ISO/IEC 42010: 2007 and IEEE Std 1471-2000)*, 2011.

[73] Michael Jarschel, Daniel Schlosser, Sven Scheuring, and Tobias Hoßfeld. An evaluation of qoe in cloud gaming based on subjective tests. In *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 330–335. IEEE, 2011.

[74] Yuxuan Jiang, Zhe Huang, and Danny HK Tsang. Challenges and solutions in fog computing orchestration. *IEEE Network*, 32:122–129, 2017.

[75] Jan Ole Johanssen, Dominic Henze, and Bernd Bruegge. A syllabus for usability engineering in multi-project courses. In *SEUH*, pages 133–144, 2019.

[76] Jan Ole Johanßen. *Continuous User Understanding in Software Evolution*. Dissertation, Technische Universität München, München, 2019.

[77] Joon-Myung Kang, Hadi Bannazadeh, Hesam Rahimi, Thomas Lin, Mohammad Faraji, and Alberto Leon-Garcia. Software-defined infrastructure and the future central office. In *2013 IEEE International Conference on Communications Workshops (ICC)*, pages 225–229, 2013.

[78] Robert Kates, William Clark, Robert Corell, Michael Hall, Carlo Jaeger, Ian Lowe, James McCarthy, Hans Joachim Schellnhuber, Bert Bolin, Nancy Dickson, et al. Sustainability science. *Science*, 292(5517):641–642, 2001.

[79] Sebastian Klepper, Stephan Krusche, Sebastian Peters, Bernd Bruegge, and Lukas Alperowitz. Introducing continuous delivery of mobile apps in a corporate environment: A case study. In *2015 IEEE/ACM 2nd International Workshop on Rapid Continuous Software Engineering*, pages 5–11. IEEE, 2015.

[80] Philippe Kruchten. The 4+ 1 view model of architecture. *IEEE software*, 12(6):42–50, 1995.

[81] Philippe Kruchten. The software architect. In *Working Conference on Software Architecture*, pages 565–583. Springer, 1999.

[82] Philippe Kruchten. What do software architects really do? *Journal of Systems and Software*, 81:2413–2416, 2008.

[83] Stephan Krusche and Lukas Alperowitz. Introduction of continuous delivery in multi-customer project courses. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 335–343, 2014.

[84] Stephan Krusche, Lukas Alperowitz, Bernd Bruegge, and Martin Wagner. Rugby: an agile process model based on continuous delivery. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, pages 42–50, 2014.

[85] Jianhua Li, Jiong Jin, Dong Yuan, Marimuthu Palaniswami, and Klaus Moessner. Ehopes: Data-centered fog platform for smart living. In *2015 International Telecommunication Networks and Applications Conference (ITNAC)*, pages 308–313, Washington, DC, USA, 2015. IEEE Computer Society.

[86] Wei Li and Sallie Henry. Object-oriented metrics which predict maintainability. *Computer Science Virginia Tech*, 1993.

[87] Zhiyong Li, Peter Mills, and John Reif. Models and resource metrics for parallel and distributed computation. *International Journal of Parallel, Emergent and Distributed Systems*, 8:35–59, 1996.

[88] J. Liebetrau and S. Grollmisch. Predictive maintenance with airborne sound analysis. *Processing Magazine*, 2017.

[89] Yuhua Lin and Haiying Shen. Cloudfog: Leveraging fog to extend cloud gaming for thin-client mmog with high quality of service. *IEEE Transactions on Parallel and Distributed Systems*, 28(2):431–445, 2016.

[90] Yutu Liu, Anne Ngu, and Liang Zeng. Qos computation and policing in dynamic web service selection. In *Proceedings of the 13th international World Wide Web Conference on Alternate Track Papers & Posters*, pages 66–73. Association for Computing Machinery, 2004.

[91] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. Fog computing: A taxonomy, survey and future directions. In *Internet of everything*, pages 103–130. Springer, 2018.

[92] Petra Maresova, Ivan Soukal, Libuse Svobodova, Ehsan Javanmardi, et al. Consequences of Industry 4.0 in business and economics. *Economies*, 6:46–60, 2018.

[93] D Margulius. Apps on the edge. *InfoWorld*, 24(21):898–909, 2002.

[94] Eva Marín-Tordera, Xavi Masip-Bruin, Jordi García-Almiñana, Admela Jukan, Guang-Jie Ren, and Jiafeng Zhu. Do we all really know what a fog node is? current trends towards an open definition. *Computer Communications*, 109:117–130, 2017.

[95] Hermann Meissner, Rebecca Ilsen, and Jan Aurich. Analysis of control architectures in the context of Industry 4.0. *Procedia CIRP*, 62:165–169, 2017.

[96] Bertrand Meyer. Applying'design by contract'. *Computer*, 25(10):40–51, 1992.

[97] Peter Mildenberger, Marco Eichelberg, and Eric Martin. Introduction to the dicom standard. *European radiology*, 12:920–927, 2002.

[98] Paul Mockapetris. Domain names - implementation and specification. Technical report, RFC Editor, 1987.

[99] Robert Monroe, Andrew Kompanek, Ralph Melton, and David Garlan. Architectural styles, design patterns, and objects. *IEEE software*, 14(1):43–52, 1997.

[100] Luc Moreau. Time-dependent unidirectional communication in multi-agent systems. *arXiv preprint math/0306426*, 2003.

[101] Mario Mustra, Kresimir Delac, and Mislav Grgic. Overview of the dicom standard. In *2008 50th International Symposium ELMAR*, volume 1, pages 39–44. IEEE, 2008.

[102] Peter Naur and Brian Randell. Software engineering: Report of a conference sponsored by the nato science committee, garmisch, germany, 7th-11th october 1968. 1969.

[103] Institute of Electrical and Electronics Engineers. Ieee standard for a software quality metrics methodology. *IEEE Std 1061-1992*, pages 1–96, 1993.

[104] Maximilian Opbacher. Adaption of an automated machine learning approach for the analysis of industrial equipment sensor data, master's thesis, technical university of munich. Master's thesis, Technical University of Munich, Munich, 2019.

[105] Lothar Pantel and Lars Wolf. On the impact of delay on real-time multiplayer games. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '02, pages 23–29. Association for Computing Machinery, 2002.

[106] David Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[107] David Parnas. On a "buzzword": hierarchical structure. In *Programming Methodology*, pages 335–342. Springer, 1978.

[108] David Parnas. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, SE-5(2):128–138, 1979.

[109] Dewayne Perry and Alexander Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, 17(4):40–52, 1992.

[110] Sebastian Peters. *MIBO – A Framework for the Integration of Multimodal Intuitive Controls in Smart Buildings*. Dissertation, Technische Universität München, München, 2016.

[111] Jon Postel. User datagram protocol. Technical report, RFC Editor, 1980.

[112] K P.Saharan and Anuj Kumar. Fog in comparison to cloud: A survey. *International Journal of Computer Applications*, 122:10–12, 2015.

[113] Han Qi and Abdullah Gani. Research on mobile cloud computing: Review, trend and perspectives. In *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, pages 195–202. IEEE, 2012.

[114] Frederick Reichheld. The one number you need to grow. *Harvard business review*, 81(12):46–55, 2003.

[115] William Riddle. *Tutorial on software design*. IEEE Computer Soc., 1980.

[116] Dmytro Rud, Andreas Schmietendorf, and Reiner Dumke. Resource metrics for service-oriented infrastructures. *Proc. SEMSOA 2007*, pages 90–98, 2007.

[117] Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012.

[118] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[119] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4):14–23, 2009.

[120] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8:14–23, 2009.

[121] Jeff Sauro and Erika Kindlund. A method to standardize usability metrics into a single score. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '05, pages 401–409. Association for Computing Machinery, 2005.

[122] Constantin Scheuermann. *A Metamodel for Cyber-Physical Systems*. Dissertation, Technische Universität München, München, 2017.

[123] Paul Schmiedmayer. A multi-scalable fog architecture in infrastructure environments, guided research, technical university of munich. Guided research, Technical University of Munich, Munich, 2018.

[124] Paul Schmiedmayer. A dynamic fog architecture in smart environments, master's thesis, technical university of munich. Master's thesis, Technical University of Munich, Munich, 2019.

[125] Stan Schneider. *The Industrial Internet of Things (IIoT)*, pages 41–81. John Wiley & Sons, Inc., 2007.

[126] Andreas Seitz. *An Architectural Style for Fog Computing: Formalization and Application*. Dissertation, Technische Universität München, München, 2019.

[127] Andreas Seitz, Dominik Buchinger, and Bernd Bruegge. The conjunction of fog computing and the industrial internet of things-an applied approach. In *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 812–817. IEEE, 2018.

[128] Andreas Seitz, Dominic Henze, Jochen Nickles, Markus Sauer, and Bernd Bruegge. Augmenting the industrial internet of things with emojis. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 240–245. IEEE, 2018.

[129] Andreas Seitz, Jan Ole Johanssen, Bernd Bruegge, Vivian Loftness, Volker Hartkopf, and Monika Sturm. A fog architecture for decentralized decision making in smart buildings. In *Proceedings of the 2Nd International Workshop on Science of Smart City Operations and Platforms Engineering*, pages 34–39. ACM, 2017.

[130] Mary Shaw and Paul Clements. The golden age of software architecture. *IEEE software*, 23(2):31–39, 2006.

[131] Mary Shaw and David Garlan. Characteristics of higher-level languages for software architecture. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1994.

[132] Mary Shaw and David Garlan. Formulations and formalisms in software architecture. In *Computer Science Today*, pages 307–323. Springer, 1995.

[133] Mary Shaw and David Garlan. Software architecture: Perspectives on an emerging discipline. *Prentice-Hall*, 1996.

[134] Mary Shaw, David Garlan, et al. *Software architecture*, volume 101. prentice Hall Englewood Cliffs, 1996.

[135] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[136] Nicolaj Siggelkow. Persuasion with case studies. *Academy of management journal*, 50(1):20–24, 2007.

[137] Jill Slay, Benjamin Turnbull, et al. The uses and limitations of unidirectional network bridges in a secure electronic commerce environment. In *Proceedings of the Fourth International Network Conference 2004 (INC2004)*, page 129. Lulu. com, 2004.

[138] Ivan Stojmenovic. Machine-to-machine communications with in-network data aggregation, processing, and actuation for large-scale cyber-physical systems. *IEEE Internet of Things Journal*, 1:122–128, 2014.

[139] Andrew Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2013.

[140] Thiago Teixeira, Sara Hachem, Valérie Issarny, and Nikolaos Georgantas. Service oriented middleware for the internet of things: A perspective. In *Towards a Service-Based Internet*, pages 220–229. Springer Berlin Heidelberg, 2011.

[141] Kip Thorne. *Black Holes & Time Warps: Einstein's Outrageous Legacy (Commonwealth Fund Book Program)*. WW Norton & Company, 1995.

[142] Luis Vaquero and Luis Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *SIGCOMM Comput. Commun. Rev.*, 44:27–32, 2014.

[143] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 20–26. IEEE, 2016.

[144] Zheng Wang and Jon Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on selected areas in communications*, 14:1228–1234, 1996.

[145] Anthony Wasserman. *Tutorial on software design techniques*. IEEE Computer Society Press, 1990.

[146] Robert Westbrook. A rating scale for measuring product/service satisfaction. *Journal of marketing*, 44:68–72, 1980.

[147] Roel Wieringa. Design science as nested problem solving. In *Proceedings of the 4th international conference on design science research in information systems and technology*, pages 1–12. Association for Computing Machinery, 2009.

[148] Roel Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.

[149] Roel Wieringa and Johannes Heerkens. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. *Requirements engineering*, 11(4):295–307, 2006.

[150] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements engineering*, 11(1):102–107, 2006.

[151] Felix Wortmann and Kristina Flüchter. Internet of things. *Business & Information Systems Engineering*, 57(3):221–224, 2015.

[152] Han Xu, Stephan Krusche, and Bernd Bruegge. Using software theater for the demonstration of innovative ubiquitous applications. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 894–897. Association for Computing Machinery, 2015.

[153] Hansong Xu, Wei Yu, David Griffith, and Nada Golmie. A survey on industrial internet of things: A cyber-physical systems perspective. *IEEE Access*, 6:78238–78259, 2018.

[154] Rerngvit Yanggratoke, Jawwad Ahmed, John Ardelius, Christofer Flinta, Andreas Johnsson, Daniel Gillblad, and Rolf Stadler. Predicting service metrics for cluster-based services using real-time analytics. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 135–143. IEEE, 2015.

[155] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78. IEEE, 2015.

[156] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: Concepts, applications and issues. In *Proceedings of the 2015 workshop on mobile big data*, Mobidata '15, pages 37–42, 2015.

[157] Robert Yin. *Case study research and applications: Design and methods*. Sage publications, 2017.

[158] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[159] Wuyang Zhang, Jiachen Chen, Yanyong Zhang, and Dipankar Raychaudhuri. Towards efficient edge cloud augmentation for virtual reality mmogs. In *Pro-*

ceedings of the Second ACM/IEEE Symposium on Edge Computing, SEC '17, pages 1–14. Association for Computing Machinery, 2017.

[160] Zibin Zheng, Yilei Zhang, et al. Cloudrank: A qos-driven component ranking framework for cloud computing. In *2010 29th IEEE Symposium on Reliable Distributed Systems*, pages 184–193. IEEE, 2010.

[161] John Zinky, David Bakken, and Richard Schantz. Architectural support for quality of service for corba objects. *Theory and Practice of Object Systems*, 3:55–73, 1997.