Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

On Providing On-demand Interoperability Solutions for the IoT

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Michael Hammerer, BSc.
Matrikelnummer 01127218

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Dr.techn. Hong-Linh Truong

Wien, 13. Dezember 2018

Michael Hammerer   Hong-Linh Truong

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

On Providing On-demand Interoperability Solutions for the IoT

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Michael Hammerer, BSc.
Registration Number 01127218

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Dr.techn. Hong-Linh Truong

Vienna, 13th December, 2018

Michael Hammerer   Hong-Linh Truong

Erklärung zur Verfassung der Arbeit

Michael Hammerer, BSc.
Edlbacherstraße 11/10, 4020 Linz

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

Wien, 13. Dezember 2018

Michael Hammerer

Danksagung

Allen voran, möchte ich mich bei meinem Betreuer Priv.-Doz. Dr. Hong-Linh Truong bedanken. Er gab mir zu jeder Zeit wertvolles Feedback und dank seiner Unterstützung konnte ich viel Wichtiges erlernen. Die harte Arbeit und Freizeit, die er investiert hat, um mir zu helfen, schätze ich sehr.

Des Weiteren möchte ich mich bei Dipl.-Ing. Lingfan Gao für die unterhaltsamen Diskussionen bedanken, die wir im Zuge unserer Diplomarbeiten hatten. Die gemeinsame Arbeitszeit war mir ein Vergnügen.

Ohne die großartige Unterstützung meiner Eltern wäre ich nicht in der glücklichen Lage in der ich mich befinde. Ich bin euch auf ewig dafür dankbar. Weiters danke ich meinen Großeltern und meinen Geschwistern Tobias und Theresa, die mich alle stets förderten und mir immer zur Seite stehen.

Es fällt mir schwer, die immense Dankbarkeit an meine Freundin Sarah in Worte zu fassen. Es weiß sicherlich niemand so gut wie du, wie viele Stunden ich in diese Arbeit investiert habe. Danke, dass du mich stets unterstützt und motiviert hast. Ohne dich hätte ich es nicht geschafft.

Zu guter Letzt möchte ich mich bei all meinen Freunden bedanken. Mit euch durfte ich schon viele unterhaltsame Stunden verbringen und ich freue mich jetzt schon auf die schöne Zeit, die uns noch erwartet.

Acknowledgements

First of all, I would like to thank my advisor Priv.-Doz. Dr. Hong-Linh Truong. He was a great support and always provided new ideas and feedback when I needed it. I very much appreciate all the hard work and time that he invested in helping me.

I would also like to thank Dipl.-Ing. Lingfan Gao for the pleasent discussions that we had while working on the thesis. I really enjoyed our cooperation and the positive spirit that he provided.

Without the support of my parents throughout my whole life, I would not have been able to achieve this milestone. Thank you for everything. Additionally, I want to thank my siblings for the support that they provided to me.

I want to express my profound gratitude to my girlfriend Sarah. You were always there for me when I needed you. I feel blessed that I can share all the great moments with you and I'm really thankful for having you.

Finally, I want to thank all of my wonderful friends for the great times that we had with eachother.

Kurzfassung

Da immer mehr *Dinge* dem Internet beitreten, wird die Interoperabilität von *Dingen* eine wichtige Angelegenheit. Eine Vielzahl an Herausforderungen entstehen wenn *Dinge* in der Lage sein sollen, mit anderen Softwarekomponenten zu interoperieren. Weil *Dinge* eine enormes Anwendungsbereich ermöglichen, sind sie höchst heterogene Aktoren. *Dinge* sind so designt, dass sie dem Anwendungskontext gerecht werden und daraus folgt, dass die von *Dingen* bereitgestellten Daten höchst verschieden sind. Dazu kommt, dass *Dinge* unterschiedlichste Technologien verwenden, da die Anforderungen an den Technologien divergieren.

Softwaresysteme, die sich das Internet der Dinge (nachfolgend IoT) zu Nutzen machen, werden Cloudservices, Netzwerkfunktionen und *Dinge* integrieren müssen. Sogenannte IoT Cloud Systems werden wichtige Ermöglicher von IoT Applikationen. Aber damit IoT Cloud Systems funktionieren, müssen deren Komponenten miteinander interoperieren. Die Diversität der *Dinge* stellt jedoch eine Herausforderung für die Interoperabilität von IoT Cloud Systems dar. Angemessene Interoperabilitätslösungen werden benötigt um die Technologie- und Datendiversität zu bewältigen, sodass *Dinge* mit IoT Cloud Systemen interoperieren können. Die Technologie- und Datendiversität an sich ist jedoch nicht die einzige Interoperabilitätsherausforderung die bewältigt werden muss. *Dinge* sind oft mobil und von ihrem Kontext und ihrer Umwelt abhängig. Wenn sich *Dinge* in der Welt bewegen oder den Kontext wechseln, werden sie IoT Cloud Systeme dynamisch betreten und verlassen müssen. Es ist daher nicht ausreichend, Interoperabilitätsprobleme mit statischen Mitteln zu bewältigen. Interoperabilitätslösungen müssen on-demand bereitgestellt werden.

Das Detektieren von Interoperabilitätsproblemen und das Finden angemessener Lösungen ist noch immer eine erschöpfende und teure Tätigkeit. Das selbe gilt für das Bereitstellen und das Anwenden solcher Lösungen. All diese Schritte sind jedoch nötig um Interoperabilitätslösungen on-demand bereitzustellen. Das Ziel dieser Diplomarbeit ist, die Zeit und den Aufwand zu reduzieren, der benötigt ist um solche Tätigkeiten durchzuführen. Durch das Modelieren von den Interoperabilitätsfähigkeiten von *Dingen*, Netzwerkfunktionen und Cloudservices mithilfe von Interoperabilitätsmetadaten, bestreben wir Interoperabilitätsprobleme zu detektieren und Lösungen vorzuschlagen. Softwarekomponenten, die als fähig angesehen werden das Problem zu lösen, werden wiederverwendet und angewendet.

Außerdem diskutieren und definieren wir die Eigenschaft der on-demand Interoperabilität und mehrere Fakturen die entscheidend sind um on-demand Interoperabilität bereitzustellen. Wir stellen Interoperabilitäts DevOps als die wichtigsten Interessensvertreter unseres Frameworks vor, da komplexe Interoperabilitätsproblem mit menschlicher Intelligenz gelöst werden müssen. Ein Proof-of-Concept Prototyp der das Framework implementiert wurde im Zuge der Diplomarbeit entwickelt. Der Prototyp wurde anschließend verwendet um die Angemessenheit des Frameworks zu evaluieren.

Abstract

As more and more *Things* join the Internet, the interoperability of *Things* becomes a major concern. Several challenges arise when *Things* should be able to interoperate with other software components. Since *Things* enable an enormous area of application, they are highly heterogeneous actors. *Things* are designed to fit the application context and as a result the data that *Things* offer varies greatly. Additionally, *Things* use diverse technologies since their requirements diverge.

Software systems that utilise the IoT will have to integrate cloud services, network functions and *Things*. Such IoT cloud systems will be major enablers of IoT applications. But in order for an IoT cloud system to function, the components of the system have to interoperate with each other. However, the *Thing* diversity presents a challenge when the interoperability of IoT cloud systems is pursued. Appropriate interoperability solutions are needed to manage the technology and data diversity, such that *Things* can interoperate with IoT cloud systems. Nevertheless, the technology and data diversity itself is not the only interoperability challenge that needs to be resolved. *Things* are often mobile and depend strongly on their context and environment. As *Things* move through the world or change contexts, they will have to dynamically enter and leave IoT cloud systems. It is therefore not sufficient, to manage the interoperability problems with static means. Interoperability solutions need to be provided on-demand.

Detecting interoperability problems and finding appropriate solutions is still an exhaustive and costly task. Same is true for provisioning and deploying such solutions. However, all of those steps are required to provide on-demand interoperability solutions. The aim of this thesis is to reduce the time and effort that is required to perform those tasks. By modeling the interoperability capabilities of *Things*, network functions and cloud services with interoperability metadata, we endeavour to detect interoperability problems and recommend solutions. Furthermore, we reuse and deploy software components that are deemed capable of solving the problem.

Additionally, we discuss and define the property of on-demand interoperability and several factors that are key to providing on-demand interoperability. We introduce Interoperability DevOps as the main stakeholders of our framework, since complex interoperability problems must be resolved with human intelligence. A proof-of-concept prototype that implements the framework was built during this thesis. The prototype was then used to evaluate the appropriateness of the framework.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

List of Figures **xvii**

List of Tables **xix**

1 Introduction **1**
1.1 Introduction 1
1.2 Motivation Scenario and Research Questions 2
1.3 Research questions 2
1.4 Contributions 3
1.5 Outline 4

2 State of the Art **7**
2.1 Overview 7
2.2 IoT cloud systems 7
2.3 IoT Interoperability 8
2.4 Interoperability in existing IoT Frameworks 11
2.5 Related Work 11
2.6 Summary 13

3 Motivation Scenarios and Use Cases **15**
3.1 Overview 15
3.2 Application-domain: INTER-IoT Seaport 15
3.3 First Scenario: Upload Camera Footage 18
3.4 Modular Interoperability Scenarios 20
3.5 Use Cases 25
3.6 Summary 37

4 Interoperability Framework **39**

4.1 Overview 39
4.2 On-demand Interoperability 40
4.3 Additional Interoperability Factors 41
4.4 Interoperability DevOps 43
4.5 Architecture and Models 48
4.6 Detecting Problems - Interoperability Check 60
4.7 Interoperability Recommendation 62
4.8 Discussion and Extensibility 66
4.9 Summary 68

5 Prototype 69

5.1 Overview 69
5.2 Prototype Architecture and the rsiHub Framework 70
5.3 Test-driven Development 72
5.4 Summary 85

6 Evaluation 87

6.1 Overview 87
6.2 Functional Evaluation 87
6.3 Performance Evaluation 97
6.4 Summary 99

7 Conclusion and Future Work 105

7.1 Conclusion 105
7.2 Future Work 107

Bibliography 109

List of Figures

1.1 Example of interoperability bridges 3

3.1 Motivation Scenario: Upload Camera Footage 19
3.2 Use case diagram 27

4.1 Interoperability DevOps Workflow 45
4.2 Framework Architecture Overview 48
4.3 SliceInformation Model 50
4.4 Interoperability Metadata Model 52
4.5 Interoperability Metadata: Protocols 53
4.6 Interoperability Metadata: Dataformats 54
4.7 Interoperability Metadata: Quality of Data 55
4.8 Interoperability Metadata: Quality of Service 55
4.9 Interoperability Metadata: Data Contract 56
4.10 Interoperability Metadata: Prototypes 56
4.11 Interoperability Software Artefact Model 58
4.12 Interoperability Bridge Model 60
4.13 Recommendation Solution Types 64

5.1 prototype architecture overview 70
5.2 rsiHub architecture taken from [Tru18] 72

6.1 Evaluation workflow 92
6.2 Screenshot of the Interoperability Service API 95
6.3 Screenshot of the Software Artefact Service API 95
6.4 Software artefact commands of pizza.js 96
6.5 Screenshot of the interoperability check with pizza.js 96
6.6 Screenshot of the interoperability recommendation with pizza.js 97
6.7 Interoperability check response time (by increasing the number of nodes) 100
6.8 Interoperability check response time (by increasing the number of metadata mismatches) 101
6.9 Interoperability recommendation response time (by increasing the number of nodes) 102

6.10 Interoperability recommendation response time (by increasing the number of metadata mismatches) 103

List of Tables

3.1 Scenario Factor: Protocol 20
3.2 Scenario Factor: Dataformat 21
3.3 Scenario Factor: Data Syntax 21
3.4 Scenario Factor: Data Contract 22
3.5 Scenario Factor: Quality 22
3.6 Scenario Factor: Quality of Data 23
3.7 Scenario Factor: Quality of Data 23
3.8 Scenario Factor: Combination 24
3.9 Scenario Factor: Data Syntax 24
3.10 rsiHub entity descriptions that are required for the use cases 25
3.11 Entity descriptions that are required for the use cases 26
3.12 UC01: Add Interoperability Software Artefact to rsiHub 28
3.13 UC02: Deploy Interoperability Software Artefact to Resource 29
3.14 UC03: Create Interoperability Bridge 30
3.15 UC04: Search Interoperability Software Artefact / Interoperability Bridge / Resource 31
3.16 UC05: Update Interoperability Metadata of Interoperability Software Artefact / Interoperability Bridge / Resource 32
3.17 UC06: Check the interoperability of a SliceInformation 33
3.18 UC07: Check the interoperability of a SliceInformation with a SliceContract 34
3.19 UC08: Get interoperability recommendations for a SliceInformation . . . 35
3.20 UC09: Get interoperability recommendations for a SliceInformation with a SliceContract 36

5.1 Testslices that are based on the seaport application domain. 74
5.2 Basic testslices that represent common problem graphs (Part 1 of 2) . . . 75
5.3 Basic testslices that represent common problem graphs (Part 2 of 2) . . . 76
5.4 Testslices that were derived from the slices of [GGH^{+}15] 77
5.5 Detailed testcase description: working slice 78
5.6 Detailed testcase description: indirect mismatch 78
5.7 Detailed testcase description: push pull problem 79
5.8 Detailed testcase description: chaining 79
5.9 Detailed testcase description: indirect mismatch 1:N 80

5.10 Detailed testcase description: M:1 81
5.11 Detailed testcase description: M:N 82
5.12 Detailed testcase description: basic circle 83
5.13 Detailed testcase description: indirect, bidirectional problem with three circles 84

6.1 VM specifications for the functional evaluation 88
6.2 VM specifications for the performance evaluation 98

CHAPTER 1

Introduction

1.1 Introduction

The *"Internet of Things"*(IoT) is the idea of an information network that includes not only computers but also *Things*, ranging from simple *Things* such as temperature sensors to highly complex *Things* such as vessels. The IoT hits it's highest potential, when *Things* can operate with a wide range of applications independent from the specific technology running on the *Thing*. This property is called interoperability and a McKinsey Study [MCB$^+$15] estimates that interoperability directly influences 40% of the potential economic value of the IoT.

With the emergence of the "Everything-as-a-Service" business model ([LW14], [DFZ$^+$15], [BFB$^+$11]), modern software applications often consist of cloud services, network functions and *Things* (also termed as resources [LNT16]). Such IoT cloud systems are major enablers of the IoT. But since *Things* are often mobile and change systems, the composition of such IoT cloud systems often changes dynamically. It is therefore important to support the dynamic and rapid provisioning of IoT cloud systems [Gao18]. But even when it's possible to provision IoT cloud systems dynamically, it is not ensured that the components of the system are interoperable. Therefore, it is necessary to also support the provisioning of on-demand interoperability solutions. The interoperability of software components and resources is however not an obvious feature. Factors that determine if two components are interoperable include but are not limited to:

- **Communication protocols and properties** since they determine if the components are able to communicate with each other
- **Data formats and syntax** determine if the components can interpret the exchanged data
- **Data rights and contracts** are important to establish an interoperable legal foundation, especially when accessing sensible information or using a component "as-a-service"

- **Quality of Service** interactions between components can fail when QoS requirements are not met
- **Quality of Data** IoT cloud systems might require certain data quality measures in order to work
- **Data semantics** determine if both components agree on the meaning of the interpreted data

Frameworks that enable the dynamic provisioning of IoT cloud systems encapsulate the cloud services, network functions and *Things* into resource slices ([TN16], [Tru18]). This thesis assumes the availability of a resources provisioning framework such as in [Gao18] to provide on-demand interoperability solutions for interoperable resource slices.

1.2 Motivation Scenario and Research Questions

A basic motivation scenario is presented in Figure 1.1. This scenario is further elaborated in Chapter 3. Consider that an alarm triggers the necessity of a resource slice ([TN16]) that sends camera data to a data storage. A resource slice includes a set of IoT units, network functions and cloud services ([TN16]). The user application, respectively the operator of the application, determines the cameras that should upload the data to a selected data storage. As the cameras have different APIs and communication capabilities, it is however not clear, whether the cameras can actually upload the video footage or not. Furthermore, if a camera is indeed not interoperable with the data storage, the interoperability problem has to be resolved. Nethertheless this could turn out to be a demanding task, since an immediate solution is not available and has yet to be developed.

The goal of this thesis is therefore to develop a framework that enables the detection of interoperability problems for IoT cloud systems. Additionally, we want to provide on-demand interoperability solution to reduce the time and effort that is necessary to solve interoperability problems, especially recurring problems.

1.3 Research questions

Based on the previously defined motivation scenario and goal, the thesis aims to answer the following research questions:

- **RQ1 Interoperability description** How can we describe the interoperability capababilities of IoT units, network functions, resources and IoT software components? What information is needed to deal with the aforementioned interoperability factors? How can we exploit the interoperability descriptions of IoT software components to reduce the effort of dealing with interoperability problems?

- **RQ2 Detecting interoperability problems** Which IoT interoperability problems can be detected based on the interoperability description of resource slice

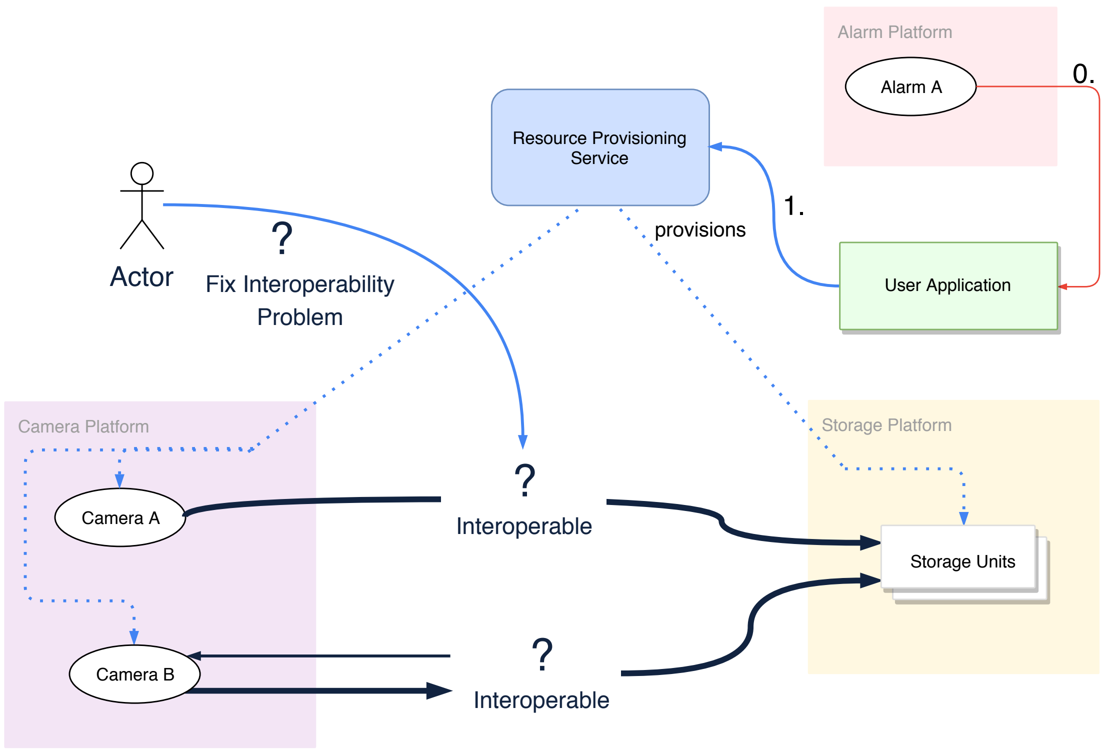


Figure 1.1: Example of interoperability bridges

components? Which problems are hard to detect? What information is necessary to describe interoperability problems?

- **RQ3 Resolving interoperability problems** What steps are necessary to create an interoperable resource slice? What entities can we utilise to reduce the effort of solving IoT interoperability problems? How can we improve the deployment process of such entities?

- **RQ4 Interoperability framework** What are the main stakeholders of our framework? How do we integrate our framework with the resource provisioning service that the framework requires? How can we evaluate our framework?

To answer the research questions, this thesis develops an interoperability framework and a prototype. The next section presents an overview of the contributions and projects that the thesis contributes to.

1.4 Contributions

We contribute a framework that supports the following features:

- modelling the interoperability capabilities of IoT resources and IoT software components with interoperability metadata
- automated detection of interoperability problems within resource slices by leveraging interoperability metadata
- recommendation of interoperability solutions based on interoperability metadata values
- integrating software artefacts that can be reutilised to solve interoperability problems
- deploying interoperability software artefacts to resources by utilising resources providers
- creating interoperability bridges by leveraging resource slices and interoperability metadata

Additionally, we contribute a prototype that implements the interoperability framework. The prototype was built in cooperation with the author of [Gao18] and uses the resource provisioning capabilities of same author's work. To this end, the prototype was built on the foundational work of [LNT16] and [TN16]. The prototype is available at `https://github.com/SINCConcept/HINC`. Additional components that are utilised by the prototype are available at `https://github.com/rdsea/IoTCloudSamples`. Some of those software artefacts were also contributed during the work on this thesis.

The work on this thesis and the created prototype contribute to the INTER-HINC project ([II18b]), which is part of the INTER-IoT project [II18a]. The work also contributes to the rsiHub framework from [Tru18]. The prototype of this thesis extends the rsiHub's prototype, which itself is based on the works of [TN16] and [LNT16].

The application-domain and scenarios of Chapter 3 are based on our industry collaboration and the Inter-Iot project. Having these realistic scenarios as foundation to our scenarios and obtaining realistic testdata was a great benefit. Additionally, the process of developing and evaluating our prototype greatly benefited from the Google Cloud Platform Research Grant (TU Wien).

Some concepts of this thesis contributed to the article "Service Architectures and Dynamic Solutions for Interoperability of IoT, Network Functions and Cloud Resources"([TGH18]) which was published in the 12th European Conference on Software Architecture, September 24-28, 2018, Madrid, Spain.

1.5 Outline

The rest of the thesis is structured as follows: The State of the Art is presented in Chapter 2 by providing important background information and related works. An application-domain, from which several scenarios and use cases are being derived, is introduced in Chapter 3. The Chapter presents the motivation of the framework. The defined use cases where used for implementing and evaluating the prototype. Chapter 4 proposes an interoperability framework that aims to enable the on-demand provisioning of interoperability solutions. Within this Chapter, the term of on-demand interoperability is discussed, followed by an introduction of supplementary on-demand interoperability layers and the concept of Interoperability DevOps. Then, the framework architecture and data models for interoperability metadata, interoperability software artefacts and interoperability bridges are provided. The Chapter is completed with descriptions of the framework's interoperability check and recommendation. The prototype that implements the interoperability framework is discussed in Chapter 5. Subsequently, an evaluation of the prototype is provided in Chapter 6. Finally, the conclusion and future work is provided in Chapter 7.

CHAPTER 2

State of the Art

2.1 Overview

We begin our discussion of the State of the Art by providing information about IoT cloud systems and how to provision such systems. Furthermore, we provide information about the resource provisioning framework that this thesis utilises to provide on-demand interoperability solutions. Thereafter, we discuss the term interoperability and it's relevance to the IoT. In doing that, we provide a definition and examine the interoperability property in more detail. Additionally, we present the role of quality measures and data contract in IoT interoperability before providing some frameworks that deal with IoT interoperability. Finally, we provide related research efforts that we organise by topics that are relevant for this thesis.

2.2 IoT cloud systems

The term IoT cloud systems is used to describe systems that incorporate *Things*, network functions and cloud services. With the growing importance of the IoT, as well as the "Everything-as-a-Service" business model, IoT cloud systems are gaining more and more importance. The composition and management of IoT cloud systems is a challenging task, as many services have to be operated and monitored and all independent elements of the IoT cloud system need to work together properly.

2.2.1 Provisioning IoT cloud systems

IoT cloud systems are highly distributed applications, especially when dealing with a wide variety of services providers. In the provisioning step of an IoT cloud system, all services need to be configured properly, so that they can eventually communicate with each other. As each software component or service might have a different API or requires a different configuration, the provisioning of IoT cloud systems is a complex task due to the heterogenity that needs to be dealt with. To deal with this complexity, [LNT16] introduced the concept of Harmonisation of IoT, Network Functions and Cloud System.

The authors of [TN16] introduce the concept of resource slices as a representation of an IoT cloud system's information model. Within resource slices, the *Things*, network functions and cloud services that are part of the IoT cloud system, are unified to resources. Based on the notion of resource slices from [TN16] and [Tru18], as well as the work on the harmonisation of IoT, network functions and cloud services from [LNT16], the article [Tru18] then introduces a framework architecture for provisioning resource slices, respectively IoT cloud systems. More work on the provisioning of IoT cloud systems can be found in [Gao18], where a proof-of-concept prototype that incorporates the architecture from [Tru18] was implemented. The author of [Gao18] also uses the term "resource ensamble" as a synonym for the term resource slice. However, the provisioned resources of a resource slice are not inherently able to properly work together, as the provisioning process solely deals with the acquisition and configuration of resources and their connectivity settings, and does not concern questions about the interoperability of resources, respectively resource slices.

2.2.2 rsiHub - Resource provisioning framework

Our framework is built upon on a resource provisioning framework. We assume that the resource provisioning framework is capable of provisioning IoT cloud systems that are defined by resource slices. While our framework does not necessarily require one specific resource provisioning service, as long as it provides the required capabilities, we use the rsiHub framework ([Tru18] and [Gao18]) for our prototype implementation.

The rsiHub framework offers one single interface for requesting and provisioning resources, called Global Management Service. The Global Management Service controls several Local Management Service through the RabbitMQ implementation of the AMQP protocol. The Local Management Services in turn use adapters to interface with resource providers ([Gao18]). The flexibility of the rsiHub allows it to integrate several resource providers. The rsiHub framework harmonises the operations that are used to control resources providers and configure the resources. Resource slices contain resources, which can be *Things*, network functions and cloud services.

2.3 IoT Interoperability

Oxford's Online Dictionary defines the word interoperability as *"the ability of computer systems or software to exchange and make use of information"* [ODO18]. This definition certainly defines the positive, working state of interoperable computer systems. However, in order to get a deeper understanding of the term interoperability, it is also necessary to understand what has to be established in order for computer systems to be interoperable and, to investigate the definition from the contraposition, what leads to systems not being interoperable. One frequently quoted and hence important work on the interoperability of systems is the Levels of Conceptual Interoperability Model by Wang et al ([WTW09])

2.3.1 Levels of Conceptual Interoperability Model

In the research literature about IoT Interoperability, the Levels of Conceptual Interoperability Model by Wang et al ([WTW09] is frequently cited. As already evident in the name of the model, it is organised in multiple interoperability levels, respectively layers. The first layer of the model, that is right above the *"no interoperability"* level, is called *"technical interoperability"* and it describes the availability of technical connections that allows the systems to exchange data. In this level, data is considered to be without structure or meaning, as this is part of levels that are above *"technical interoperability"*. The next level, syntactic interoperability or level 2, is established when the systems *"have an agreed protocol to exchange the right forms of data in the right order, but the meaning of data elements is not established"*, [WTW09].

Once two (or more) systems can exchange data with an agreed protocol and the structure of the data is understood by both systems, the next level, semantic interoperability, is about reaching a common understanding of the data semantics. Further knowledge of the systems' semantics is exchanged in the next level, pragmatic interoperability, where the meaning of the systems' contexts is also known by all parties. Thus, two systems that are pragmatically interoperable, are not only able to agree on the semantics of the exchanged terms, but can also utilise the meaning of the information context.

Furthermore, systems that are able to deal with dynamic contexts, thus changing the meaning of the own context and reacting to changes of the counterpart's context, are categorised in level 5, dynamic interoperability. Systems that are dynamically interoperable *"are able to re-orient information production and consumption based on understood changes to meaning, due to changing context as time increases"*, [WTW09]. The highest level of the model, is the level of conceptual interoperability, where the *"interoperating systems are completely aware of each others information, processes, contexts, and modeling assumptions"*, [WTW09]. The authors are however not certain, if conceptual interoperability can actually be reached.

From the aforementioned definition of [ODO18] we can deduce that the interoperability of computer systems or software can fail when either the ability to exchange information or the ability to make use of information is impaired. To conclude the Levels of Conceptual Interoperability Model, the ability to exchange information is impaired when the technical or physical data exchange is broken, or when no syntactic agreement about the data exchange is in place. On the other hand, to correctly make use of another system's information, the meaning of data and context is necessary as otherwise there is the risk of failing to make the correct use and hence not being interoperable. Additionally, interoperability can fail if the meaning of the context changes and systems fail to react to such changes.

While it is is extremely difficult and challenging to make systems self-interoperable towards other systems, our effort is to enable on-demand interoperability. The benefit of enabling on-demand interoperability is that the systems do not have to be self-interoperable, as interoperability will be established by the operators of the systems, depending on the current interoperability situation and context of the systems. Hence, when the circumstances with respect to any interoperability level change and interoperability is not given anymore, on-demand interoperability solutions can repair the ability of the systems to interoperate. On-demand interoperability solutions can therefore be used to establish de facto dynamic interoperability.

2.3.2 The role of Quality of Service, Quality of Data and Data Contracts in IoT interoperability

As we deal with IoT cloud systems and the on-demand provisioning of interoperability solutions, Data Contracts are also topics of concern, as they influence a system's ability to exchange and make use of another systems information. Same is true for Quality of Service (QoS) and Quality of Data (QoD) metrics.

In [TGC+11], an abstract model for Data Contracts is presented. The authors of [BT17] propose a contract-aware IoT framework that allows the monitoring of QoD and QoS metrics. The main contract categories of both works consist of

- **Quality of Data** mentioned QoD metrics are the *completeness* and *conformity* with respect to a negotiated data model, *average message age*, *average message currency*, *consistency* and *interpretability*. However, the appropriate metrics that should be applicated, depending on the domain of the data asset.
- **Quality of Service** mentioned QoS metrics are *availability* and *reliability*. Same as for QoD metrics, the actual QoS metrics depend on the capabilities of the resource.
- **Data Rights** contains information about whether the *derivation*, *collection* or *reproduction* of the data is allowed. Additional properties determine if the data is qualified for *noncommercial use* only and whether the provider requires *attribution* or not.
- **Regulatory Complience** is represented by a list of regulatory acts that protect or apply to the data asset.
- **Pricing Model** contains information about the financial effort it takes to aquire the asset or resource. Valuable properties of pricing models are the *price*, *currency* and the *unit* that is sold in the process.
- **Control and Relationship** contains information about the *jurisdiction*, the *warranty*, *liability* and *indemnity* that is tied to the data asset or resource.
- **Purchasing Policy** is a contract subcategory that holds information about the *contract termination*, *shipping* and *refund*

As mentioned before, the IoT hits its highest potential when interoperability is established. IoT cloud systems can then be composed with resources from many different "Everything-as-a-service" platforms [LW14]. In our view, properties from the above listed categories are important to achieve this goal. To enable interoperable "Everything-as-a-service" resources in IoT cloud systems, data contracts and quality metrics need to be considered.

2.4 Interoperability in existing IoT Frameworks

To provide a short overview of the fastly growing amount of standards, projects and technologies that concern the IoT, we list a couple of important technologies, frameworks and projects. Taking into account the lists of [Pos18] and [GGH+15], there are currently around sixty prominent protocol standards for the IoT. Some widely used application protocol standards are: MQTT, AMQP, CoAP, STOMP, HTTP, XMPP. Examples for dataformats that are widely utilised in the IoT are JSON, CSV, YAML, XML, CBOR, Apache AVRO. Frequently used dataformats for representing semantic data are: JSON-LD, RDF. Other frameworks and projects that deal with or are related to IoT interoperability are:

- **oneIoTa** is a web-based tool that aims to define interoperable device data models for the IoT, [OCF18].
- **INTER-IoT** is a research project that *"aims to design, implement and test a framework that will allow interoperability among different Internet of Things (IoT) platforms"*, [II18a]. The approach of the INTER-IoT project is discussed in [FSP+18].
- **IoTivity** *"is an open source software framework enabling seamless device-to-device connectivity to address the emerging needs of the Internet of Things"*, [IoT18b]. The architectural goal of IoTivity is however to implement a new standard ([IoT18a]). The goals of IoTivity and the proposed framework of this thesis are therefore diverging.

Since many IoT frameworks are at an early stage, it is sometimes hard to figure out what each framework is striving for. Nevertheless, many frameworks push towards IoT interoperability and a lot more deal with the IoT as such. However, we were not able to locate a framework that deals with on-demand interoperability.

2.5 Related Work

We organise the related work into different categories that we deem important for our work. However, as on-demand interoperability is a fairly recent topic, most of the investigated research papers did not directly relate to this thesis. To this end, the main effort of the research literature concerning IoT interoperability is about semantic interoperability, or more precisely about how to reach a commonground for data semantics in the IoT ([JAK+16], [GS16], [SBK+17] and [AOK+15] among many others).

2.5.1 On-demand Interoperability

An on-demand SOA-based multiprotocol translater is proposed in [DED17]. While the effort to dynamically translate different protocols is similar to our approach in the sense that it is also done dynamically, our approach strives for reusing resources such as the proposed translater. The focus of our work is on attaching interoperability information to concrete resources such that those resources can be used from a higher abstraction level. On this abstraction level we detect interoperability problems and solve them by reusing appropriate software components, such as the multiprotocol translator. The benefit of this approach is that interoperability problems that go beyond protocol issues can be considered. In contrast to [DED17], we aim for the dynamic development of interoperability solutions. More precisely, adding new capabilities to our framework would require integrating additional resources during runtime rather than changing the framework itself. With that being said, integrating the proposed multiprotocol translater from [DED17] into our framework would certainly help solving a lot of interoperability problems related to protocols.

The article [TD15] calls for a more coherent software layer in IoT cloud systems to simplify the development of IoT cloud applications and unleash the potential of the IoT. The envisioned systems allow for dynamic changes in resources. To guarantee that the parts of the system are interoperable, it is important to provide dynamic interoperability solutions. Static interoperability means are not sufficient to solve dynamic interoperability demands that such systems create.

According to [Mil15], it is unlikely that a protocol and data format that the whole community agrees upon will emerge, due to the fast spread of IoT. This leads to an interoperability gap which needs to be closed by using other principles than standardisation. The article [Mil15] also states that interoperability is essential for reaching the potential of IoT. In our view, this work further stresses the importance of interoperability in the IoT, especially the need for on-demand interoperability.

The paper [JAK$^+$16] proposes several approaches to establish semantic interoperability. The goal is to connect vertical IoT silos, such that applications can use the data and resources of different IoT platforms accordingly. The authors note that *"even if today's IoT systems are willing to expose their data and resources to others their semantically incompatible information models become an issue to dynamically and automatically interoperate as they have different descriptions or even understandings of resources and operational procedures"*. They later present the H2020 symbIoTe project that achieves semantic interoperability with semantic mapping and by dynamically rewriting SPARQL queries. While the core approach also depends on metadata, the goal is slightly different to ours. Our focus is on establishing on-demand interoperability for IoT cloud systems by improving the ability to create interoperable slices, in contrast to establishing semantic interoperability of IoT platforms. We therefore don't require technologies of the semantic web, but reuse software components to achieve interoperability.

2.5.2 Interoperability Check

The authors of [GPS16] developed a tool based on the concept of model-driven engineering that helps developers with detecting interoperability problems, such that found problems can be manually resolved by developers. The detection of problems using lightweight interoperability models might be of interest for our work. However, in contrast to our approach, this work does not aim to dynamically reuse, create and provide interoperability solutions. We view the work of [GPS16] as complementary to our approach.

2.5.3 Interoperability Recommendation

One core component of the article "An Architecture for Interoperable IoT Ecosystems" [SBK$^+$17] are marketplaces that allow consumers and providers to exchange resources and data. Such a marketplace could also be used for the interoperability recommendation, where the marketplace is queried for resources and software components that are capable of solving on-demand interoperability problems. Additionally, the Interoperability Software Artefacts (Section 4.5.4) and Interoperability Bridges (4.5.5) could also be offered and monetised within a marketplace. Apart from that, the proposed architecture also strives for interoperability between IoT-platforms to allow for cross-platform and cross-domain applications. To achieve this, the proposed work also relies on semantic web technologies and it emphasises the importance of semantic interoperability.

2.6 Summary

In contrast to mentioned research efforts, our work is about on-demand interoperability solutions for the IoT and the dynamic development of interoperabiliy solutions. The thesis is strongly bound to the topic of dynamic provisioning of IoT cloud systems (background provided in Section 2.2). As the provisioned systems are highly dynamic, the interoperability problems that come with it, also have a dynamic nature and therefore require on-demand solutions. The consequence is that our work has to incorporate dynamic techniques to solve the problems. That being said, the scientific literature about dynamic interoperabilty problems in the IoT is fairly sparse at this point in time.

CHAPTER 3

Motivation Scenarios and Use Cases

3.1 Overview

This chapter presents a set of scenarios based on an application of the INTER-IoT project ([II18a]) that motivates the work in this thesis. After defining the most important entities of the application-domain and explaining the dynamic aspect of the seaport, the first scenario provides a more detailed insight into the problem at hand. Furthermore, several scenarios of interoperability layers and problems within the domain of the seaport are introduced. Finally, this chapter provides use cases that have been created with the scenarios in mind.

3.2 Application-domain: INTER-IoT Seaport

In order to reach a high level of application, the entities of the interoperability framework is very generic. This statement also applies to the resource provisioning service that the interoperability framework depends on. To get a better grasp of the idea and the framework's capabilities, it is important to tie the framework to real world scenarios that can later be used to verify the prototype. Additionally, those real world scenarios help with explaining and understanding the abstractions of the generic framework, as it is often easier to apply concrete thinking rather than abstract thinking.

Our application-domain is based on a modern, intelligent seaport. The entities of the application-domain are equipped with IoT technologies. We then introduce motivating scenarios that are based on this application-domain. The entities that we need for our seaport scenarios are:

- **Port Communication System (PCS)** manages all the communication within the seaport and is used by the Port Control System to manage entities within the port. The *Port Communication System* itself exposes APIs that allow external entities to request current tracking data (for instance information about the approaching *Vessels*). For matters of simplicity, we do not further discuss the capabilities of the Port Control System as the scenarios below do not necessarily require it.

- **Vessels** dynamically enter and leave the port. *Vessels* have to communicate and interoperate with the *Port Communication System.* Exchanged information contains for instance (but is not limited to) the current location and the status of the *Vessel* as well as commands from the *Port Communication System.*

- **Access Gates** are equipped with sensors that detect when a car enters or leaves the port, so that the amount of cars in the seaport can be tracked.

- **Weather Sensors** do not only send their data to the *Port Communication System*, but can also be requested by external parties.

- **Cameras** have APIs that allow to access live camera footage as well as the most recent video footage that is in the camera's memory.

- **Alarm Sensors** (e.g. Smoke Detectors) send the alarm information to the *Port Communication System*

Additionally, we have external stakeholders and systems that will interact with the port. Those entities are:

- **Fire Brigade** might require video footage in case of a fire alarm

- **Cloud Data Storage Platforms** as data sinks for video footage

- **Logistics Companies** that request tracking data from the PCS in order to update schedules.

- **Weather Analytics Systems** that request weather data from the port.

While in a real port there would be additional entities to manage, we try to keep the application-domain and scenarios as compact as possible. Since this thesis is about on-demand interoperability solutions, we define the dynamic aspects of our application-domain in the next subsection.

3.2.1 Dynamic aspects of the application-domain

Dynamic provisioning and *on-demand interoperability* are key aspects when dealing with interoperability problems. In this thesis, the term *on-demand interoperability* means that interoperability is established with dynamic techniques in opposition to static ways of creating interoperability solutions. Such static ways of solving interoperability problems could be standardisation or by creating a solution during the development phase of a project, contrary to it's operation phase. The questions that immidiately arise are:

1. Why can't interoperability problems be solved with standardisation? If every entity uses the same standardised protocol for instance, then the interoperability problem would clearly be solved.
2. Why can't interoperability problems not be solved when designing and implementing the software project? After all, making components interoperable is a common task when developing software and a core skill of software developers.

To answer the questions and to make a case for on-demand interoperability, consider following scenario within the previously defined port. Consider that vessels will have to communicate with the PCS in order to safely enter, leave and manouver within the port. As vessels can be from any country, manufacturer or manufacturing year, they might also use different communication technologies. It would be very positive if there was one communication technology that every vessel uses, a silver bullet of communication technologies. Nevertheless, such a standard is currently not available and it does not seem like one will emerge in the near future. And even if a standard technology would soon emerge, there are still a lot of vessels that do not incorporate this standard so there are still interoperability problems that need to be taken care of. Another reason why standardisation will not solve communicative interoperability problems in the IoT is that a wide variety of standards simply exist because of the wide variety of demands. While one standard might satisfy the demand of low energy consumption with the cost of low bandwith, another standard might enable a high bandwidth but lead to a higher energy consumption. The area of application and technical capability of the *Thing* determines the demands on the standard and subsequently justifies which standard will be used. There is no silver bullet.

If it's not possible to solve the interoperability problems regarding to communication technologies with one single standard, isn't it possible to achieve the interoperability of all vessel communication technologies during the development phase of the Port Communication System?
This approach is unfortunately not sufficient either, because the problem domain is too big. As mentioned above, vessels can have any communication technology. Considering that any vessel that exists may at one point have to interact with the port, it is not possible to deal with all those cases during the development phase. When also considering other interoperability problem factors, not only the communication technology, the problem domain becomes even bigger and the task of solving such issues with static means becomes

even more impossible than it already is. And even when agile software development techniques are being applied, the development circles are too long for creating solutions to dynamic interoperability problems, as problems might need to be solved as fast as possible.

In conclusion, we must support solving interoperability problems during runtime, with interoperability solutions being developed in short development circles and solutions are best provisioned dynamically. On-demand interoperability is an important feature, but currently there is a lack of solutions. Furthermore, an extra benefit of the dynamic provisioning of interoperability solutions is that scalability and elasticity can easily be achieved.

While there are frameworks available that provide means for dynamic provisioning of resources ([Gao18], [Tru18]), they do currently not deal with interoperability. Our motivation is to provide dynamic mechanism that improve the way to deal with interoperability problems based on a framework that is already capable of dynamically provisioning resources.

3.3 First Scenario: Upload Camera Footage

To get a better grasp of the system view, consider the following scenario: In the port, a fire sensor or security door triggers an alarm that is sent to the Port Control via the Port Communication System. To analyse what caused the alarm, video footage provides important information. However, cameras might have different APIs or capabilites and some cameras might store video footage only locally. To prevent the risk of losing valuable information and to give the fire brigade a chance to analyse the situation while they are on their way, all cameras that are close to the source of the alarm should upload the last five minutes of video footage that was recorded before the incident happened, to a cloud storage. Again, as cameras might have different APIs and capabilites, it might be necessary to deal with each camera in it's own way, making them interoperable with the cloud storage. Figure 3.1 provides an overview of the situation. For this system scenario we have the following platforms and applications:

- Camera Platform
- Alarm Platform
- Data Storage Platform
- User Application

We described in Chapter 1 that our framework depends on a resource provisioning service. While this can be any framework that provides the resource provisioning capabilities, we use the rsiHub framework ([Tru18] and [Gao18]). For our scenario, the most essential services of the rsiHub framework are the Global Management Service as main communication interface and some Local Management Service that manage the resources, respectively the platforms. Since rsiHub's provisioning techniques are meant to be flexible, the ensemble of resources could as well look differently and resources do not

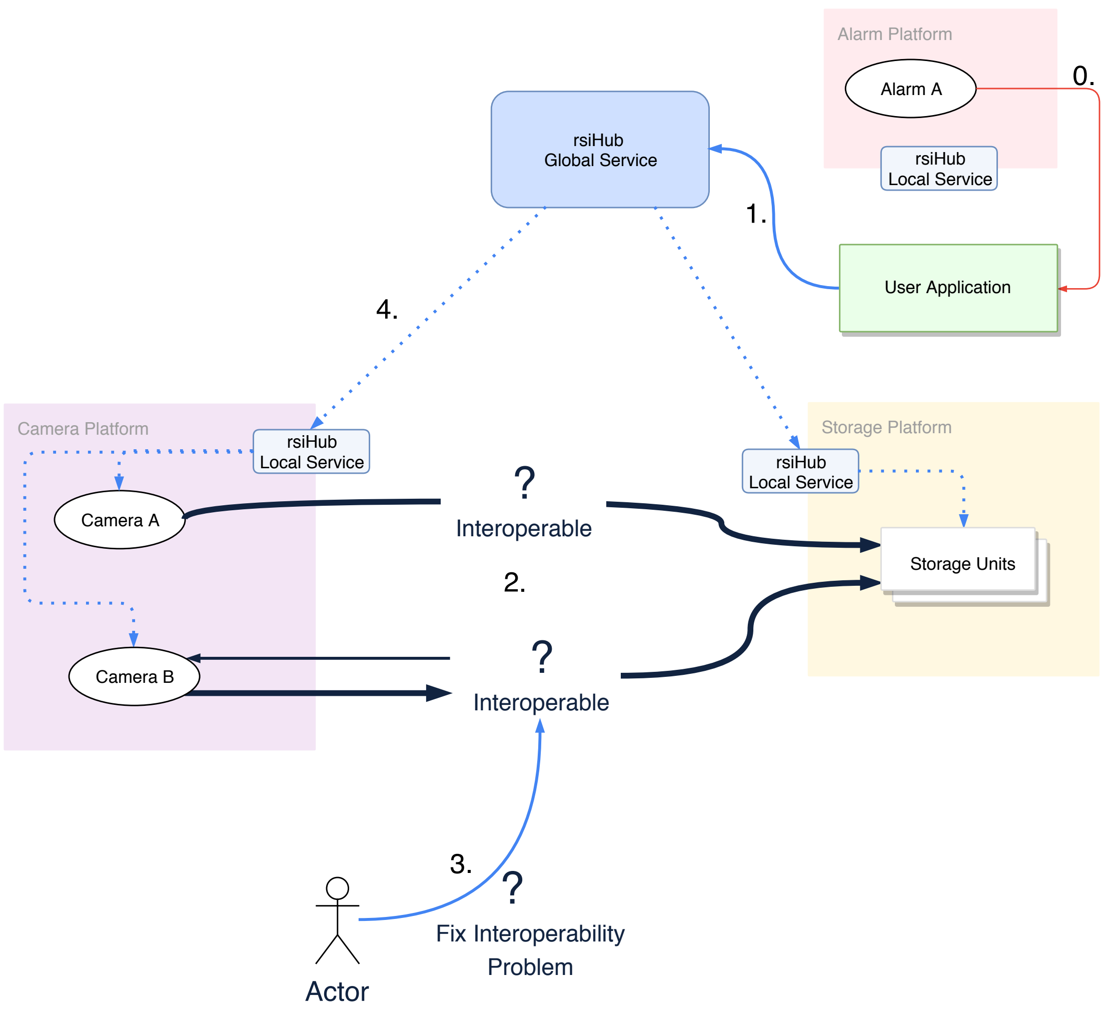


Figure 3.1: Motivation Scenario: Upload Camera Footage

have to be within a platform, but can also be run locally. For the scenario, we assume that rsiHub's services are already running.

The step that initiates the scenario is marked with **0.**, an alarm is triggered and it is sent to the user application. Either the user application itself or an operator of same application then uses the rsiHub Global Service to create an uplink from the relevant cameras to the data storage platform. However, because of the varying camera models and APIs, it is not clear, if such an uplink can be created for each camera. One problem might for instance be that camera A pushes the data to a destination while camera B requires the destination to actively request the data, so the data needs to be pulled from camera B, but getting this information requires detailed investigation of the requested resources.

It's not clear if the cameras and the data storage platform are interoperable. Therefore we have to determined if said components are interoperable, and if they are not, some Actor has to fix those interoperability problems before rsiHub provisions the resources in the last step. The successful end state would then be that all resources are provisioned and that each requested camera uploads the video footage to the data storage platform.

3.4 Modular Interoperability Scenarios

As seen in Chapter 2, interoperability factors can be categorised within several levels. The interoperability levels therefore also play an important role for this thesis and it is also important to identify interoperability factors in scenarios. This section provides several fine-grained scenarios where each scenario focuses on one particular interoperability factor. But of course, interoperabiliy problems often belong not just to one isolated interoperability level, but rather spread accross multiple interoperability levels and therefore concern multiple interoperability factors. Thus, the scenarios of this section can be seen as modules that could also be combined to more complex scenarios or interoperability problems. All of the following scenarios are based on the components of the previously defined application-domain of the INTER-IoT seaport.

3.4.1 Factor: Communication protocol

As already described, when a vessel wants to enter the Port, it has to establish communication with the Port Control, such that the vessel can receive the commands of the Port Control. The communication protocol of the vessel depends entirely on the preferences of it's manufactorer, as there is not just one protocol standard that must be used. Depending on the actual communication protocol of the vessel entering the port, the Port Control either can or can not communicate with the vessel. If there is a disparity in communication protocols, then this gap has to be closed by dynamically allocating an instance that is capable of mediating between the protocols of vessel and Port Control. For instance, while the Port Control uses the MQTT protocol, the vessel can only communicate using the AMQP protocol.

Factor	Protocol
Example	
From	MQTT
To	AMQP
Dynamic Aspect	Although protocol mediations are rather static transformations, the benefit of making this transformation dynamically is that computing resources are only consumed when really needed.

Table 3.1: Scenario Factor: Protocol

3.4.2 Factor: Data format and syntax

Staying at the vessel scenario from the previous factor, even when protocol communication is established, it is not guaranteed that the exchanged information is interpreted correctly, as the vessel might use a different dataformat than the Port Control. Therefore, the dataformat factor is also essential when establishing a working data communication channel.

One example would be that the Port Control uses the JSON dataformat, while the vessel uses CSV. Thus, a mediating instance would have to transform the JSON data to CSV and vice versa, in order to establish a working data exchange.

Factor	Dataformat
Example	
From	CSV
To	JSON
Dynamic Aspect	Same as for protocols, the benefit of not running these transformation services all the time is that computing resources are saved.

Table 3.2: Scenario Factor: Dataformat

An additional scenario that would fall into the dataformat factor are standardised data models. Assume that both the vessel and the Port Communication System use JSON as dataformat, but the way that the actual data is structured differs. While the vessel uses a data model that confirms to the oneIoTa standard, the PCS uses the InterIoT standard. An instance that knows how to translate both standards is required as mediator.

Factor	Standardised Data Models of IoT Platforms
Example	
From	oneIoTa
To	InterIoT JSON-LD
Dynamic Aspect	

Table 3.3: Scenario Factor: Data Syntax

3.4.3 Factor: Data Contract

Jurisdiction: Assume that the cameras of the port can also be requested by external parties. For instance, in case of a fire emergency, the most recent camera data is aquired by the fire brigade, so that the firefighters can prepare for the situation while they are on their way to the source of the fire. To aquire the video material, assume that the camera has to store the data to a cloud storage facility that the fire brigade uses. Furthermore, assume that for legal reasons, it is required that the camera data can only be stored within the European Union. Thus, the cloud storage facility, as well as other intermediate entities of the communication chain, have to be located within the European Union, so that there is no legal interoperability problem.

Data Rights: As mentioned in the application-domain, the port possesses weather sensors of which the data can be acquired by external parties. The assumption for this kind of interoperability problem is that the port requires the data consumers to only utilise the data for non-commercial usage.

Pricing: The Port Control keeps track of the vessels and additionally possesses access gate sensors that can be used to keep track of the cars entering and leaving the port. Those statistics can be used by external logistics companies to optimise delivery plans to and from the port and to quickly reschedule in case of vessel delays. As the tracking data has economic value, the port charges a price for consuming the data. Logistics companies that do not want to pay for the data should therefore not be able to acquire the resource.

Factor	Data Contract
Example	
From	Collection = any
To	Collection = true
Dynamic Aspect	In a dynamic application, data contract issues need also be considered among all parties, such that no violations to legal interoperability appear.

Table 3.4: Scenario Factor: Data Contract

3.4.4 Factor: Quality of Service

Reliability: The access gate sensors that detect when a car enters or leaves the port have a reliability measure. Assume that data consumers, like logistics companies as described above, require a certain reliability value in order to trust and use the data that they want to consume. Therefore, only access gates that possess a reliabity value that is above the threshhold should be connected to the data consumers.

Factor	Quality of Service
Example	
From	Reliability = any
To	Reliability >= 90
Dynamic Aspect	In a dynamic application, unreliable components can easily be removed or replaced for more reliable components.

Table 3.5: Scenario Factor: Quality

Message Delivery Frequency: Consider that, due to performance reasons, an external consumer of the weather data only wants to receive one update per minute. The weather sensors on the other hand, send two updates per minute. Thus, the consuming application and the sensors are not interoperable on the quality of service level.

Factor	Quality of Service
Example	
From	Message Delivery Frequency = 1s
To	Message Delivery Frequency = 2s
Dynamic Aspect	It is important that the resources limitations are considered, as otherwise the resource might deny it’s service.

Table 3.6: Scenario Factor: Quality of Data

3.4.5 Factor: Quality of Data

Precision: The data consumer that wants to acquire weather measurements requires that the temperature sensors measure with a precision of 0.1°C.

Average Measurement Age: To assert that the aquired data is relevant within the current time frame, the logistics companies require that the average measurement age of the aquired data does not exceed a certain threshold.

Factor	Quality of Data
Example	
From	Average Measurement Age = 5s
To	Average Measurement Age = 2s
Dynamic Aspect	The age of a measurement can play a crucial role for the value of the data.

Table 3.7: Scenario Factor: Quality of Data

3.4.6 Multi-factored Interoperability Problems

Interoperabitility problems do not occur purely seperated by factor, they can also be combined which creates more complicated solution efforts. Consider the vessel scenario where the vessel uses AMQP and CSV an the Port Communication System uses MQTT and JSON. Multi-factored problems are another reason why dynamic interoperability is important.

Factor	Protocol & Dataformat
Example	
From	MQTT, CSV
To	HTTP, JSON
Dynamic Aspect	One benefit of a dynamic approach is that resources can be chained and combined to dynamically create more powerful transformations, leading to a smaller amount of basic transformations that can be combined for multiple problems.

Table 3.8: Scenario Factor: Combination

3.4.7 Higher Factors: General Data Transformations and Data Semantics

The status updates from the vessel put too much stress on the network because a lot of unnecessary information is sent, resulting in large messages and further leading to low throughput of important information within the Port Communication System. As a lot of the status information is not considered anyways, the information of the status updates should be filtered, resulting in only forwarding relevant information to the Port Communication System. An instance of a software component that only forwards the important information should therefore mediate between vessel and Port Communication System.

Factor	Data Transformation
Example	
From	"<keyA>:<value>"
To	"<keyB>:{<keyC>:<value>}"
Dynamic Aspect	These kind of problems are greatly varying because the data models of data producers and consumers are not (and can not easily be) standardised. When dealing with dynamic systems and components, such problems can not be solved during the development phase. It's also not easily possible to solve such problems purely with algorithmic measures. Solving such problems dynamically, by deploying a custom-tailored transformation service, is a good approach of dealing with such problems.

Table 3.9: Scenario Factor: Data Syntax

3.5 Use Cases

With the previously defined scenarios, we created use cases that are key to our framework. The use cases are generic and can be applied to a wide range of scenarios and applications. Since the interoperability framework enhances rsiHub, the use cases contain some key definitions of rsiHub. Following table should provide a better understanding of those rsiHub elements and how they can be tied to the application-domain:

Name	**Description**	**Port Scenario Examples**
Resource	A running computing instance, network function or IoT entity	An IoT-vessel, a Node-RED instance
SliceInformation	A (requested) set of connected Resources that has not been provisioned yet	The PCS connected to a message broker connected to a vessel
Resource slice	A provisioned SliceInformation	See SliceInformation. The difference is that the Resources are up and interacting with each other

Table 3.10: rsiHub entity descriptions that are required for the use cases

Additionally, the use cases already contain some elements from the interoperability framework that will be introduced in Chapter 4. Table 3.11 should give an idea about those elements, more details will be provided in Chapter 4.

The preconditions that are assumed to be true for each of the use cases are as follows:

- rsiHub Global Management Service is online
- rsiHub Local Management Services are online
- SliceInformation, SliceContract and Interoperability Metadata definitions follow the correct syntax

Name	**Description**	**Port Scenario Examples**
Software Artefact	A piece of software that can be executed by a Resource	A Node-RED flow that can transform csv to json
Interoperability Metadata	Information about the interoperability capabilities	An Interoperability Metadata instance that describes that the Node-RED flow can transform csv to json
Interoperability Software Artefact	A Software Artefact with Interoperability Metadata definition	See Software Artefact and Interoperability Metadata
Interoperability Bridge	A SliceInformation that can be used for interoperability purposes, together with an Interoperability Metadata definition	Two Software Artefacts that are connected to each other to make a more complex transformation
SliceContract	A set of conditions that a resource slice needs to fulfill	A SliceContract instance with the information that the resource slice will be used for commercial purposes

Table 3.11: Entity descriptions that are required for the use cases

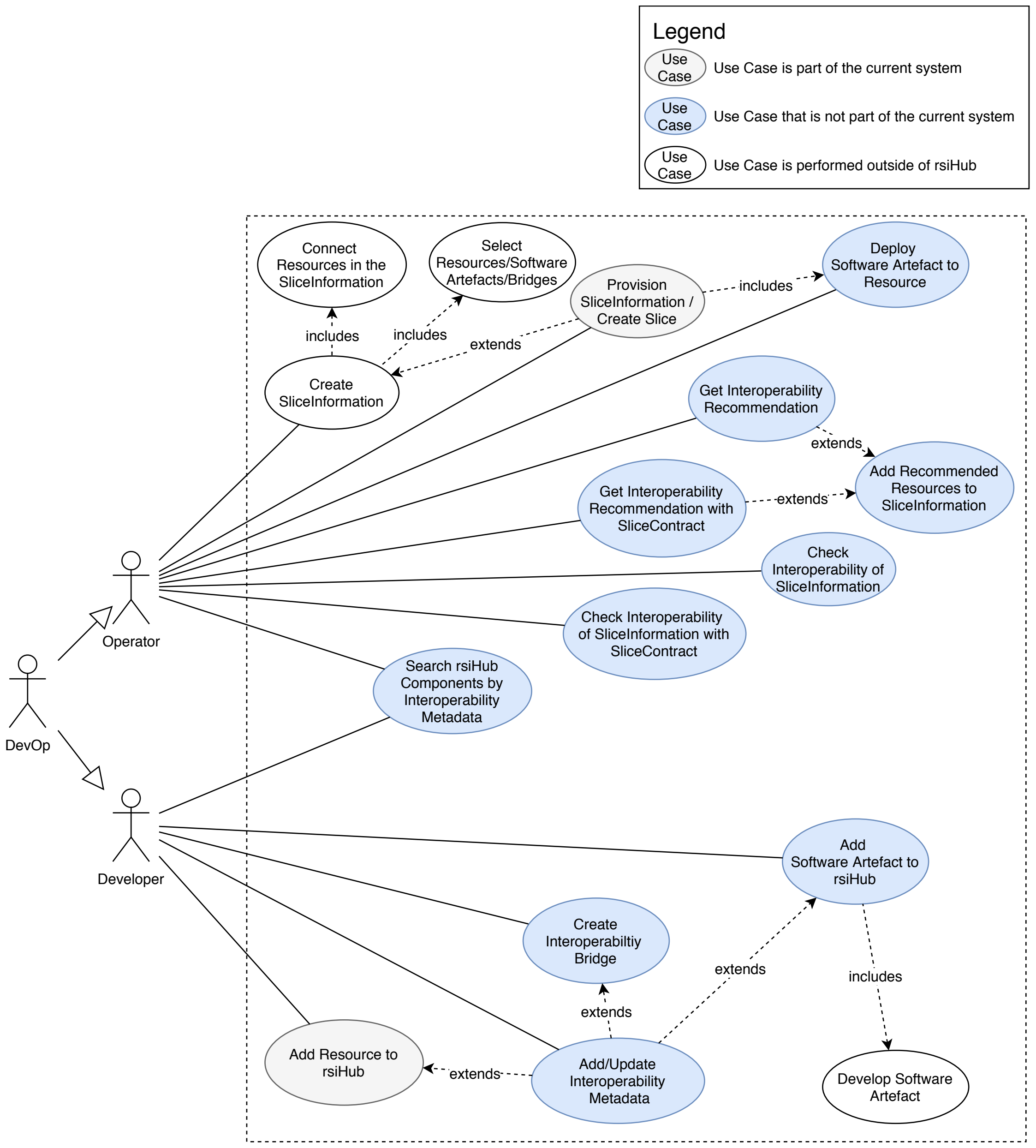

Legend
Use Case
Use Case is part of the current system
Use Case
Use Case that is not part of the current system
Use Case
Use Case is performed outside of rsiHub
Connect Resources in the SliceInformation
Select Resources/Software Artefacts/Bridges
Provision SliceInformation / Create Slice
Deploy Software Artefact to Resource
includes
includes
includes
extends
Create SliceInformation
Get Interoperability Recommendation
extends
Add Recommended Resources to SliceInformation
Get Interoperability Recommendation with SliceContract
extends
Check Interoperability of SliceInformation
Check Interoperability of SliceInformation with SliceContract
Operator
Search rsiHub Components by Interoperability Metadata
DevOp
Developer
Add Software Artefact to rsiHub
Create Interoperabiltiy Bridge
extends
includes
extends
Add Resource to rsiHub
extends
Add/Update Interoperability Metadata
Develop Software Artefact

Figure 3.2: Use case diagram

ID	UC01
Title	Add Interoperability Software Artefact to rsiHub
Problem Statement	A developer wants to add a Software Artefact to rsiHub that can be used to solve interoperabiltiy problems.
Preconditions	• The Software Artefact is already developed and available (either locally or remotely)
Main Scenario	1. The developer uses rsiHub's command line client to add an Interoperability Software Artefact 2. The developer provides the path or URL of Interoperability Software Artefact 3. The developer provides the Interoperability Metadata or the path to a json document that contains the Interoperability Metadata of the Interoperability Software Artefact 4. The developer provides the execution environment of the Interoperability Software Artefact 5. The developer provides a name for the Interoperability Software Artefact 6. rsiHub creates a new Interoperability Software Artefact holding the provided information. If the Software Artefact was a local file, it is replicated in rsiHub's cloud storage.
Successful End States	1. The Interoperability Software Artefact has been added to rsiHub. 2. The developer receives a message that the Interoperability Software Artefact has been added

Table 3.12: UC01: Add Interoperability Software Artefact to rsiHub

ID	UC02
Title	Deploy Interoperability Software Artefact to Resource
Problem Statement	An operator wants to deploy an Interoperability Software Artefact to a Resource
Preconditions	• The Interoperability Software Artefact is available in rsiHub • A Resource that can execute the Interoperability Software Artefact is provisioned and available in rsiHub
Main Scenario	1. The operator uses rsiHub's command line client to deploy the Interoperability Software Artefact to the Resource 2. The operator provides the ID of the Interoperability Software Artefact 3. The operator provides the ID of the Resource 4. rsiHub's command line client deploys the Interoperability Software Artefact to the Resource
Successful End States	1. The Interoperability Software Artefact has been deployed to the Resource. 2. The Interoperability Software Artefact is running on the Resource.

Table 3.13: UC02: Deploy Interoperability Software Artefact to Resource

ID	UC03
Title	Create Interoperability Bridge
Problem Statement	An operator or developer wants to create an Interoperability Bridge from a previously created SliceInformation that solved an interoperability problem, such that the Interoperability Bridge can be used for similar problems.
Preconditions	none
Main Scenario	1. The operator or developer uses rsiHub's command line client to create the Interoperability Bridge 2. The user provides the path to a json document that contains the SliceInformation of the Interoperability Bridge 3. The user provides a json String or the path to a json document that contains the Interoperability Metadata of the Interoperability Software Artefact 4. The developer provides a name for the Interoperability Bridge 5. rsiHub creates a new Interoperability Bridge holding the provided information
Successful End States	1. The Interoperability Bridge is available in rsiHub.

Table 3.14: UC03: Create Interoperability Bridge

ID	UC04
Title	Search Interoperability Software Artefact/Interoperability Bridge/Resource (using Interoperability Metadata)
Problem Statement	An operator wants to search an Interoperability Software Artefact/Interoperability Bridge/Resource that can then be used to solve an interoperability problem. To search the component, the operator wants to use search attributes that are related to interoperability issues (Interoperability Metadata).
Preconditions	none
Main Scenario	1. The operator or developer uses rsiHub's command line client to search for Interoperability Software Artefact/Interoperability Bridge/Resource using Interoperability Metadata 2. The user provides a json string or the path to a json document that contains the query. The query is a valid MongoDB query.
Successful End States	1. The operator receives a list of Interoperability Software Artefacts/Interoperability Bridges/Resources that satisfy the query.

Table 3.15: UC04: Search Interoperability Software Artefact / Interoperability Bridge / Resource

ID	UC05
Title	Update Interoperability Metadata of Interoperability Software Artefact/Interoperability Bridge/Resource
Problem Statement	An operator or developer wants to change the information regarding interoperability issues (Interoperability Metadata) of an Interoperability Software Artefact/Interoperability Bridge/Resource.
Preconditions	none
Main Scenario	1. The operator or developer uses rsiHub's command line client to update the Interoperability Metadata of the Interoperability Software Artefact/Interoperability Bridge/Resource 2. The user provides the ID of the Interoperability Software Artefact/Interoperability Bridge/Resource 3. The user provides a json string or the path to a json document that contains the updated Interoperability Metadata 4. rsiHub updates the Interoperability Metadata of the Interoperability Software Artefact/Interoperability Bridge/Resource
Successful End States	1. The Interoperability Metadata of the Interoperability Software Artefact/Interoperability Bridge/Resource changed. The change is visible in rsiHub.

Table 3.16: UC05: Update Interoperability Metadata of Interoperability Software Artefact / Interoperability Bridge / Resource

ID	UC06
Title	Check the interoperability of a SliceInformation
Problem Statement	An operator wants to check if a SliceInformation contains any detectable interoperability problems before creating a resource slice from the SliceInformation
Preconditions	none
Main Scenario	1. The operator uses rsiHub's command line client to check the interoperability of the SliceInformation 2. The operator provides the path to a json document that contains the SliceInformation that will be checked
Successful End States	1. The operator receives a list of interoperability problems 2. For each interoperability problem, the operator receives information about what interoperability metadata is involved in the problem 3. For each interoperability problem, the operator receives information about what components of the SliceInformation are involved in the problem

Table 3.17: UC06: Check the interoperability of a SliceInformation

ID	UC07
Title	Check the interoperability of a SliceInformation with a SliceContract
Problem Statement	An operator wants to check if a SliceInformation contains any detectable interoperability problems before creating a resource slice from the SliceInformation. The SliceInformation should also be validated against a SliceContract that the operator provides.
Preconditions	none
Main Scenario	1. The operator uses rsiHub's command line client to check the interoperability of the SliceInformation 2. The operator provides the path to a json document that contains the SliceInformation that will be checked 3. The operator provides the path to a json document that contains the SliceContract
Successful End States	1. The operator receives a list of interoperability problems 2. For each interoperability problem, the operator receives information about what interoperability metadata is involved in the problem 3. For each interoperability problem, the operator receives information about what components of the SliceInformation are involved in the problem 4. The operator receives a list of contract violations 5. For each contract violation, the operator receives information about what component violates the contract 6. For each contract violation, the operator receives information about what aspect of the SliceContract is violated.

Table 3.18: UC07: Check the interoperability of a SliceInformation with a SliceContract

ID	UC08
Title	Get interoperability recommendations for a SliceInformation
Problem Statement	An operator wants to get recommendations that suggest how the interoperability of a SliceInformation can be improved (also called Interoperability Recommendation)
Preconditions	none
Main Scenario	1. The operator uses rsiHub's command line client to get recommendations for the SliceInformation 2. The operator provides the path to a json document that contains the SliceInformation for which the recommendations should be gathered
Successful End States	1. For each interoperability problem, the operator receives either a recommendation that would solve the problem or the information that no recommendation is available for the respective problem 2. For each recommendation, the operator receives information about what changes to the SliceInformation will solve the problem 3. For each recommendation, the operator receives information about the Interoperability Software Artefact/Interoperability Bridge/Resource that will solve the problem.

Table 3.19: UC08: Get interoperability recommendations for a SliceInformation

ID	UC09
Title	Get interoperability recommendations for a SliceInformation with a SliceContract
Problem Statement	An operator wants to get recommendations that suggest how the interoperability of a SliceInformation can be improved. The recommendations should also satisfy a SliceContract that the operator provides.
Preconditions	none
Main Scenario	1. The operator uses rsiHub's command line client to get recommendations for the SliceInformation 2. The operator provides the path to a json document that contains the SliceInformation for which the recommendations should be gathered 3. The operator provides the path to a json document that contains the SliceContract
Successful End States	1. For each interoperability problem, the operator receives either a recommendation that would solve the problem or the information that no recommendation is available for the respective problem 2. No recommendation does violate the SliceContract 3. For each recommendation, the operator receives information about what changes to the SliceInformation will solve the problem 4. For each recommendation, the operator receives information about the Interoperability Software Artefact/Interoperability Bridge/Resource that will solve the problem.

Table 3.20: UC09: Get interoperability recommendations for a SliceInformation with a SliceContract

3.6 Summary

In this chapter we introduced an application-domain that is based on the INTER-IoT seaport and then discussed the dynamic aspects that the seaport implies. From the application-domain, we derived several scenarios that serve as motivation for our framework. The first scenario should provide an overview of the involved systems and the problem situation in general. The scenarios that followed take specific properties of interoperability problems into consideration. Finally, we used our scenarios to derive use cases that are later used when implementing an evaluating the prototype.

CHAPTER 4

Interoperability Framework

4.1 Overview

As we have already shortly described in the scenarios of the previous chapter, we need on-demand interoperability solutions for dealing with dynamic interoperability problems (Section 4.2). The interoperability of multiple systems can be classified within the Levels of Conceptual Interoperability Diagram ([WTW09]). On-demand interoperability calls for additional interoperability factors and requirements (Section 4.3), that complement the Levels of Interoperability Conceptual Diagram. While the algorithmic components of this framework focus on the interoperability factors up to the corresponding Level 2, more difficult problems that incorporate higher factors require the introduction of Interoperability DevOps (Section 4.4).

One goal of this framework is to reduce the effort of dealing with interoperability problems and increase the efficiency of Interoperability DevOps by supporting them in their tasks. To achieve this goal, the central idea of this framework is to use interoperability metadata in order to reuse resources and software artefacts that are able to act as interoperability bridges (Section 4.5). Additionally, interoperability metadata is used to assist the Interoperability DevOps with finding interoperability problems (Section 4.6) and recommended solutions to those problems (Section 4.7).

In order to enable dynamic provisioning of on-demand interoperability solutions, this framework supplements the work on rsiHub, a framework that allows for the dynamic provisioning of resource ensembles. Finally, a discussion about the interoperability framework and thoughts about the extensibility of the framework conclude this chapter (Section 4.8).

4.2 On-demand Interoperability

In Chapter 3, we discussed problems that arise due to the dynamic aspects of the seaport application-domain (Section 3.2.1. This chapter aims to give a broader view and description of on-demand interoperability. In detail:

- In what situations is on-demand interoperability needed?
- What key features define on-demand interoperability?
- What are additional benefits of establishing on-demand interoperability?

When looking at the seaport scenarios from a general perspective, the essential traits of an application that make on-demand interoperability an indispensable property are:

- **need for interoperating with unknown entities:** The domain of the entities that have to interoperate with the application is either unknown, too big or too vague to define. In the port scenario, the communication technology domain of the vessels that enter the port was too big to define.
- **time constrained need for solutions:** It is important that an solution to the interoperability problem is found within a time constraint that is too short to solve the problem within the development phase of the application. In the port scenario, vessels can not wait until an interoperable reiterated version of the Port Communication System is online.
- **interoperability problems appear during runtime:** The source and destination of the interoperability problem do not allow a system shutdown. It is necessary to create a new, interoperable communication channel. In the port scenario, it is not possible to shutdown the Port Communication System and start a new, interoperable version of the Port Communication System.

On-demand interoperability is on the other hand established when those essential requirements are met. Key features that define the property of on-demand interoperability are:

- software components run as own instances/services, independent from the interoperability problem's source and destination system
- interoperability software components are strongly decoupled from the interoperability problem's source and destination system
- interoperability software components are developed in short iterations, independent from interoperability problem's source and destination system

- interoperability software components can be terminated once the interaction between source and destination is completed
- ideally, interoperability software components can be stored and reused (to reduce the time to solve a problem)

Establishing on-demand interoperability not only satisfies the previously mentioned, required traits that lead to the need of on-demand interoperability, but it also bears additional advantages. Those extra benefits are:

- **allows for increased scalability/elasticity:** new instances of interoperability software components can easily be spawned and destroyed as they run independently from the problem's source and destination systems
- **fine-grained interoperability components:** the single goal of interoperability software components is to mediate between the problem's source and destination system, respectively two technology standards
- **allows for fast development circles for interoperability solutions:** fine-grained, independent components are easier and faster to develop and complex interoperability problems can be solved with quick, responsive iterations
- **complex interoperability problems can be solved by combining smaller interoperability components** small instances of interoperability software components, that only mediate between two standards can be combined to mediate on multiple interoperability dimensions
- **enhanced interoperability evolution:** when saving such interoperability components, it reduces the effort to solve an interoperability problem.
- **increased ability to interact with new standards and legacy systems:** interoperability software components that mediate between such standards and systems can easily be created and instantiated

The Levels of Conceptual Interoperability Model from [WTW09] still applies when on-demand interoperability is enabled. In fact, dealing with interoperability problems on-demand calls for some complementary interoperability factors that we will introduce in the following section.

4.3 Additional Interoperability Factors

A great foundation to understand interoperability from a greater perspective can be gained from the Levels of Conceptual Interoperability Model by Wang et al ([WTW09]). While working on this thesis, some additional interoperability factors that are not directly mentioned in [WTW09] but are important for the interoperability framework have been considered. In our view, these additional factors complement and extend the Levels of Conceptual Interoperability Model.

Quality of Service(QoS) and Quality of Data(QoD) might implicitely be included in the Levels of Conceptual Interoperability Model, but they are not explicitely mentioned. QoD and QoS are important features in the field of distributed systems. As IoT cloud systems are distributed systems, these two factors are important to our framework.

To explain the importance of those two factors, consider a fictional situation where we have an IoT cloud system with two components that are connected with each other. The first component is an IoT alarm sensor that monitors the safety gate of a machine. The IoT alarm sensor is connected to the control system of the factory. Assume that the factory control system and the IoT sensor are semantically interoperable as otherwise the factory control system would not be able to react to the alarms. Since the IoT sensor monitors a safety gate, it has to send the data consistently to the factory control systems. If it fails to do so, the factory control system assumes that the safety gate is breached and reacts accordingly. Accordingly, when the IoT sensor is prone to lags because it's QoS is insufficient, the two components are not interoperable even though they are semantically interoperable.

As for the QoD factor, assume that another IoT sensor periodically takes measurements and sends the latest measurement to the factory control system. Assume that when the sensor value reaches a critical value, the port control systems has only a short time window within it has to trigger another event. The factory control system therefore requires that the sensor measurements are taken within a time frame of 5 seconds after the previous measurement. Additionally, assume that QoS is not an issue. If the IoT sensor takes a measurement every 30 seconds but sends the latest data every 2 seconds, the factory control system assumes everything is correct. However, a change of events might not be sensed quickly enough. In that case, the factory control system fails to react and therefore the two components are not truly interoperable.

Additionally, we introduce the Data Contract factor. This factor has no corresponding level in the Levels of Conceptual Interoperability Model as the model considers only technical aspects of interoperable systems and the Data Contract factor deals with business and legal concerns. In more detail, the Data Contract factor needs to be enabled such that the interoperation of two systems has a negationated legal set of rules as foundation for the interaction. Thus, if in contrast, a legal foundation can not be established, the two systems have to be considered unable to interoperate, even if they are interoperable on a technical Level.

The central approach of the interoperability framework is to define interoperability metadata to describe the capabilities of software components. Based on the interoperability metadata the parts of framework detect interoperability problems and construct recommendations. The Levels of Conceptual Interoperabiliy Model influenced the definition of interoperability metadata and also outlined a reasonable limit for the automated detection of problems and recommendation of solutions. That being said, this work on detection and recommendation mainly focusses on the interoperability factors up to Level 2, as well as the previously defined additional factors. Semantic interoperability problems as well as problems that require complex data syntax transformations are hard to detect and solve automatically. Hence, our framework envisiones a new group of stakeholders that deal with problems that need to be solved with human intelligence. We call these stakeholders Interoperability DevOps.

4.4 Interoperability DevOps

So far, the key stakeholders and main users of the interoperability framework were viewed as the creators and operators of resource ensembles, respectively resource slices. But viewing the operators of resource slices as key stakeholders has a shortcoming when solving interoperability problems where on-demand interoperability is required. To elicitate the need of Interoperability DevOps, we have to start the discussion with applications where on-demand interoperability is required. This framework, as an supplement of the rsiHub framework, aims to enable on-demand interoperability with all it's benefits. One part of establishing on-demand interoperability is the capability to dynamically provision interoperability solutions, which is in part already possible with rsiHub and resource slices, and can be done by the operator of the resource slice. But, in order to provision the interoperability solution, the software component that acts as the interoperability solution has to be acquired at first. To simplify this step, one goal of this framework is to describe interoperability solutions with metadata and additionally to make them reusable by utilising the interoperability metadata. However, there are interoperability problems where a tailored interoperability solution is necessary, especially from data syntax level (Level 2) upwards. The reason for this circumstance is, that complex model transformations can not be properly described with metadata. A formal language would be necessary to describe such transformations accurately. Even if the transformation are described with a formal language, it be would extremely complex or even impossible to apply such transforming interoperability solutions automatically and correctly. In fact, there even exists a formal language to describe the data model transformations: the source code of the software component that performs the data model transformation. Thus, applying such model transformation automatically and in a correct way, is as complex as learning an AI how to develop working, problem-solving software. Therefore, in order to also deal with problems of such a complexity, Interoperability DevOps are needed.

However, there are also alternatives to the Interoperability DevOps stakeholder group. Another possible concept would be that interoperability experts with software development skills solve problems for the creators and operators of resource slices. In such a strict division of workforce, operators and interoperability developers would coexist and not interfere with the task of one another. What speaks against this approach, is that the interoperability developer would certainly lack vital information and decision power that only the operators have. Nonetheless, the information and decision power might be needed in order to develop or provision an appropriate solution. While it would still be possible that the operator informs the developer about the interoperability problems in an ongoing communication, such a division of workforce, knowledge and power would lead to a communication overhead that would not only increase errors, but also lead to additional costs in time and finance. Therefore, we view the concept of Interoperability DevOps as more favourable and profitable.

We define Interoperability DevOps as *operators with skills in developing software and expertise in the interoperability of software systems and technologies*. To get a better grasp of the Interoperability DevOps concept, we looked at the tasks that Interoperability DevOps would need to perform. In our view, important Interoperability DevOp tasks are:

- Create, Deploy and Operate Slices
- Create, Read, Update, Delete software artefacts, interoperability bridges and resources
- Search/Browse software artefacts, interoperability bridges and resources
- Detect interoperability problems
- Solve interoperability problems
  - Automatically with Recommendations based on the interoperability metadata
  - Manually by browsing existing resources and software artefacts and adding them to the slice
  - Manually by creating a new software artefact and adding it to the slice

An Interoperability DevOp Workflow can be seen in Figure 4.1.

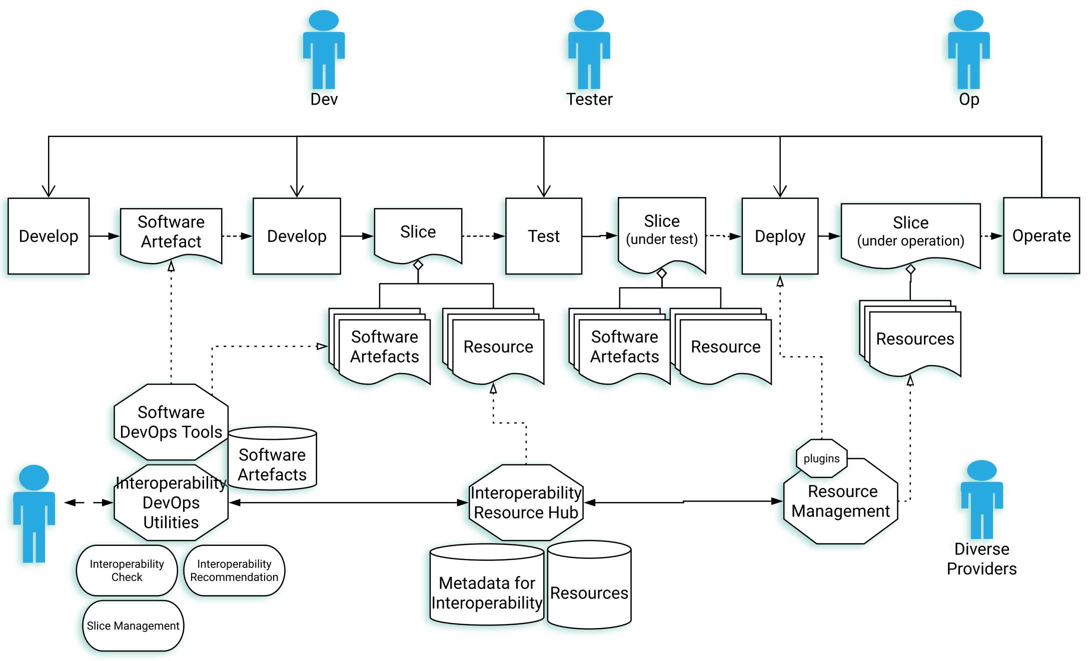


Figure 4.1: Interoperability DevOps Workflow

4.4.1 Create, deploy and operate resource slices

Resource slices are the core entities for running resource ensembles. As DevOps are operators by definition, Interoperability DevOps are also responsible for creating the SliceInformation that leads to a deployed respectively provisioned resource slice. Creating a SliceInformation is the first task in every Interoperability DevOp workflow. To create a SliceInformation, an Interoperability DevOp has to select and add resources, software artefacts and interoperability bridges to the SliceInformation and define which components should be connected. After the SliceInformation has been created (and interoperability issues have been resolved) the resource slice can be created and operated.

4.4.2 Create, read, update, delete interoperability software artefacts, interoperability bridges and resources

Interoperability DevOps will have to manage the main components of the interoperability framework that will be used to deal with interoperability problems. Especially important when solving interoperability problems are interoperability software artefacts and interoperability bridges, specifically creating such entities within the framework. When these components are created within the framework, they need to have assigned values for interoperability metadata, such that the interoperability metadata can be used for searching components, for detecting Interoperabiltiy problems and for recommending solutions.

4.4.3 Search/Browse interoperability software artefacts, interoperability bridges and resources

Interoperability DevOps must be able to efficiently access and search interoperability software artefacts, interoperability bridges and resources by their capabilities regarding interoperability, such that subsequently those components can be added to the SliceInformation.

We defined our interoperability metadata to describe the capabilities of software artefacts, interoperability bridges and resources with respect to the interoperability of components. One intention of interoperability metadata is to make search efforts of Interoperability DevOps easier.

4.4.4 Detect interoperability problems

In order to solve interoperability problems, one has to detect interoperability problems. Like a compiler that checks if the source code of a program contains any errors, Interoperability DevOps need to find as many interoperability problems as possible before provisioning the resource slice. That way, the temporal and financial costs of provisioning an erroneous resource slice can be avoided.

4.4.5 Solve interoperability problems

When a resource slice contains an interoperability problem, the problem needs to be solved. This is the very key task of Interoperability DevOps. We indentified several techniques that Interoperability DevOps can apply when solving interoperability problems. Although the goal and outcome of each technique is the same, the techniques themself are quite different from each other. The techniques of solving interoperability problems are:

1. **Automatically with recommended solutions:** with the appropriate descriptions of a components interoperability capabilities and the knowledge of what causes the interoperability problem, a solution can be recommended by an algorithm. The Interoperability DevOp can then assess the solution and provision the resource slice

2. **Manually by reusing interoperability components:** This technique is similar to the previous one except that the Interoperability DevOp herself/himself is searching and adding existing resources, interoperability software artefacts and interoperability bridges to the resource slice
3. **Manually by developing a new interoperability component:** If no suitable interoperability component is available to solve the interoperability problem, the Interoperability DevOp has to develop a new software artefact or interoperability bridge that can subsequently be added to the resource slice. In this sense, developing a new interoperability component can either mean to develop the software from scratch and then add it to the framework, or to simply add existing software to the interoperability framework.

4.4.6 Interoperability DevOp tasks and the interoperability framework

All those aforementioned tasks can also be performed without the interoperability framework. The Interoperability DevOps could alternatively gather all the computing resources, network functions, *Things* and cloud services themselves, provision and wire up all components manually. Interoperability problems can be debugged when some connection is not working and appropriate mediating services can be developed to handle those problems. However, it is still preferable to use the interoperability framework, because it increases the efficiency of getting these tasks done. It shortens the time to provision a resource slice, to find interoperability problems and solutions to eliminate those problems. And time-efficiency is key when the time to find and deploy an interoperability solutions is limited and on-demand interoperability is required.

The goal of our framework with respect to Interoperability DevOps is therefore to assist them as good as possible in executing the tasks and reduce the time that is needed to perform the tasks. One key aspect of achieving this goal is to reuse components and solutions to solve problems. Instead of *"solve a problem once, apply the solution once"* our approach is to *"solve the problem once, apply the solution often"*. To make the components and solutions reuseable, the component's description of it's interoperability capabilites is essential. Our framework defines and uses sophisticated interoperability metadata to describe these capabilities. In order for Interoperability DevOps to fully utilise our framework, having good knowledge about the interoperability metadata is key. After thoroughly defining and discussing the reasons, goals and stakeholders of this interoperability framework, the upcoming sections of this chapter focus more on the technical aspects of the framework, starting with the framework's architecture and models in the very next section.

4.5 Architecture and Models

4.5.1 Interoperability Framework Architecture

Figure 4.2 shows the architecture of our framework. The *Software Artefact Service* manages software artefacts and stores the interoperability metadata for the artefacts. The *Software Artefact Service* is also capable of deploying software artefacts to resources. To deploy the software artefact, the *Software Artefact Service* requires a service that is either capable of provisioning resources on demand or capable of providing an already running resource to the *Software Artefact Service*. The *Software Artefact Service* then takes the "ingress access point" of the resource that is provided by the resource provisioning service. The ingress access point is then used to connect to the resource and deploy the software artefact to the resource.

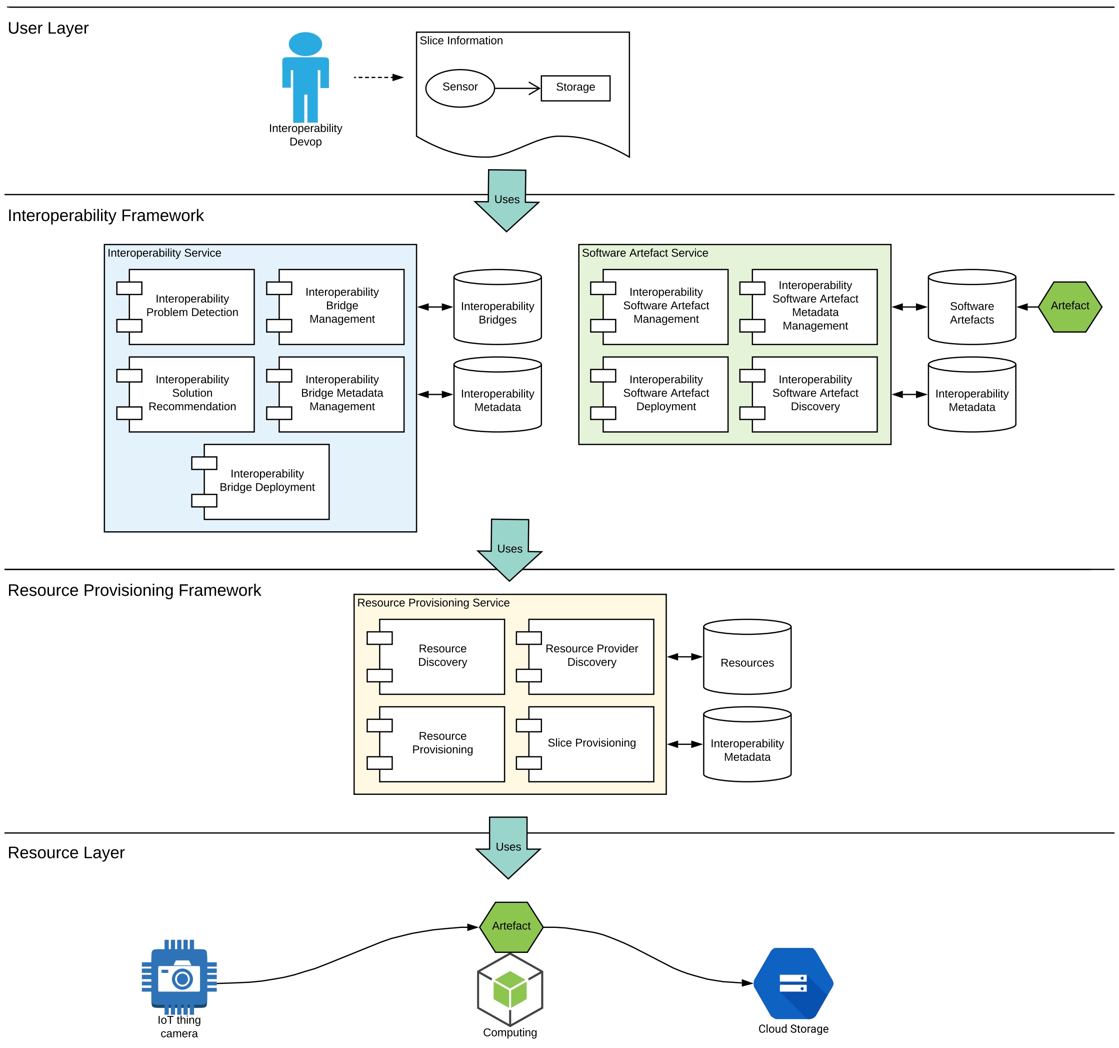

User Layer
Interoperability Devop
Slice Information
Sensor
Storage
Uses
Interoperability Framework
Interoperability Service
Interoperability Problem Detection
Interoperability Bridge Management
Interoperability Solution Recommendation
Interoperability Bridge Metadata Management
Interoperability Bridge Deployment
Interoperability Bridges
Interoperability Metadata
Software Artefact Service
Interoperability Software Artefact Management
Interoperability Software Artefact Metadata Management
Interoperability Software Artefact Deployment
Interoperability Software Artefact Discovery
Software Artefacts
Interoperability Metadata
Artefact
Uses
Resource Provisioning Framework
Resource Provisioning Service
Resource Discovery
Resource Provider Discovery
Resource Provisioning
Slice Provisioning
Resources
Interoperability Metadata
Uses
Resource Layer
Artefact
IoT thing camera
Computing
Cloud Storage

Figure 4.2: Framework Architecture Overview

The *Interoperability Service* manages interoperability bridges and performs the interoperability check and interoperability recommendation on SliceInformations. Both interoperability check and interoperability recommendation can be performed with or without considering a SliceContract. In order to perform the interoperability check and recommendation, the framwork user has to send a SliceInformation to the *Interoperability Service*. The SliceInformation is an entity or document that describes a resource slice and therefore contains a set of resources. In addition to the resources, a SliceInformation can also contain interoperability software artefacts and interoperability bridges. A set of user-defined connections determine how the user would like the components to work together (this does not mean that it is actually possible, as the interoperability of connected components needs to be checked first).

For the interoperability check and recommendation to work, the *Interoperability Service* requires that all components of a SliceInformation have an assigned interoperability metadata instance. This requirement also applies to components that will be considered for the interoperability recommendation. Components that can be considered for an interoperability recommendation are interoperability software artefacts, interoperability bridges and resources. Same as for the *Software Artefact Service*, the *Interoperability Service* requires a resource provisioning service that is capable of managing resources. For the interoperability recommendation, the resource provisioning service is required to have a search API, such that resources can be accessed based on an interoperability metadata query. The deployment of interoperability bridges is equal to the provisioning of SliceInformations. We assume that the resource provisioning service is capable of provisioning SliceInformations.

4.5.2 SliceInformation Model

Resource slices are used to provision IoT cloud systems consisting of resources such as IoT, network functions and cloud services. We assume that the resource provisioning service that our interoperability framework is built upon uses resource slices. To detect interoperability problems and recommend solutions, the Interoperability Service also has work deal with resource slices, as interoperability problems are tied to resource ensembles. Thus, we have to define the information model of resource slices that our framework uses. Subsequently, we assume that the resource provisioning service that the interoperability framework depends on uses the same information model to describe resource slices. Our SliceInformation model can be seen in Figure 4.3.

The core elements of the SliceInformation model are resources, interoperability bridges and interoperability software artefacts. However, since these SliceElements do usually not run individually, but communicate with each other, a SliceInformation must also hold information about which elements should be connected with each other. A SliceInformation can thereby be seen as a graph where resources, interoperability bridges and interoperability software artefacts represent the vertices and user-defined connections act as the edges. The SliceInformation instance is then used to create a resource slice that represents an IoT cloud system.

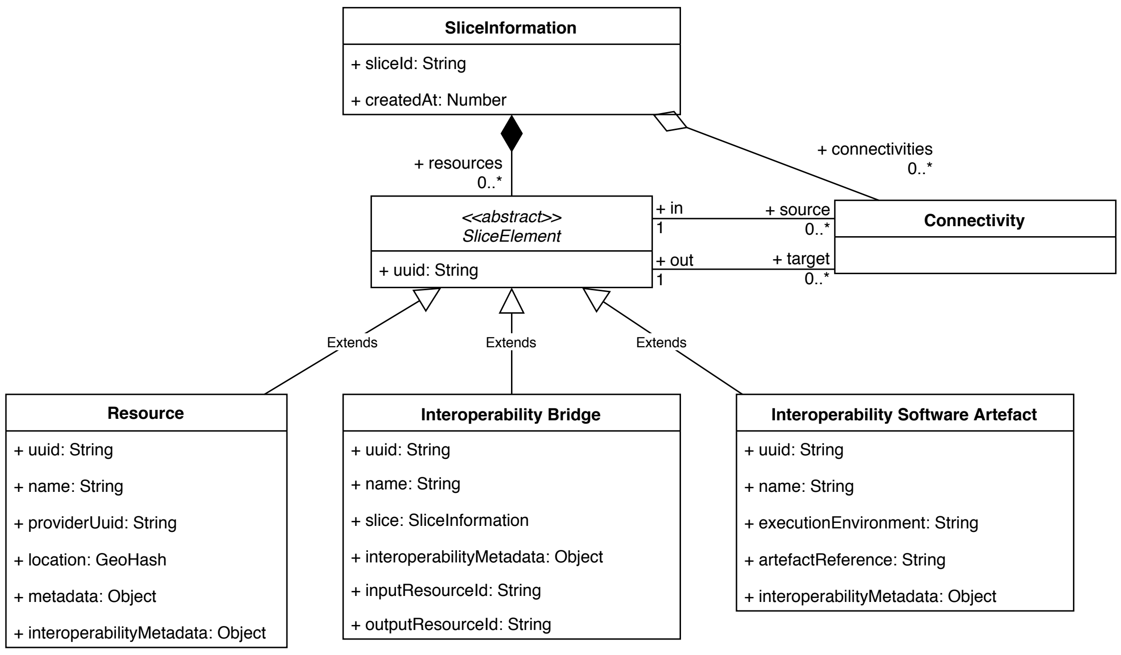


Figure 4.3: SliceInformation Model

The prototype of our framework uses documents to represent and store SliceInformation instances. The connections are defined by the creator of the SliceInformation. It is easily possible that two components are wired together are not interoperable per se, but should interoperate for the application to work. Listing 4.1 shows an example of a SliceInformation document.

4.5.3 Interoperability Metadata

The Interoperability Metadata builds the very foundation of our framework and is it utilised to dynamically reuse software components for solving interoperability problems. It is also important when detecting interoperability problems. The main information model of the Interoperability Metadata can be seen in Figure 4.4.

For describing the interoperability metadata model, we destinguish the entities of the model based on the position in the information model tree. At first we discuss the main entities that are close to the root. Afterwards, the entities in the leafs of the information model (Prototype, Data Contract, etc.) will be described in more detail. The main entities or categories of our Interoperability Metadata are

- Inputs
- Outputs
- Resource

We chose those three groups because they model the different capabilities that a SliceCom-

```
{
  "sliceId": "valencia_intop_01_protocol",
  "resources": {
    "pcs": {
      ...
      "metadata": { ...
      },
      "source": ["broker_to_pcs"],
      "target": ["pcs_to_broker"]
    },
    "vessel": {
      "metadata": { ...
      },
      "source": ["broker_to_vessel"],
      "target": ["vessel_to_broker"]
    },
    "broker": {
      "metadata": { ...
      },
      "source": ["pcs_to_broker", "vessel_to_broker"],
      "target": ["broker_to_vessel", "broker_to_pcs"]
    }
  },
  "connectivities": {
    "pcs_to_broker": {
      "in": {"label":"pcs"},
      "out": {"label":"broker"}
    },
    "broker_to_vessel": {
      "in": {"label":"broker"},
      "out": {"label":"vessel"}
    },
    "vessel_to_broker": {
      "in": {"label":"vessel"},
      "out": {"label":"broker"}
    },
    "broker_to_pcs": {
      "in": {"label":"broker"},
      "out": {"label":"pcs"}
    }
  },
  "createdAt": 1534059245
}
```

Listing 4.1: Example of a SliceInformation

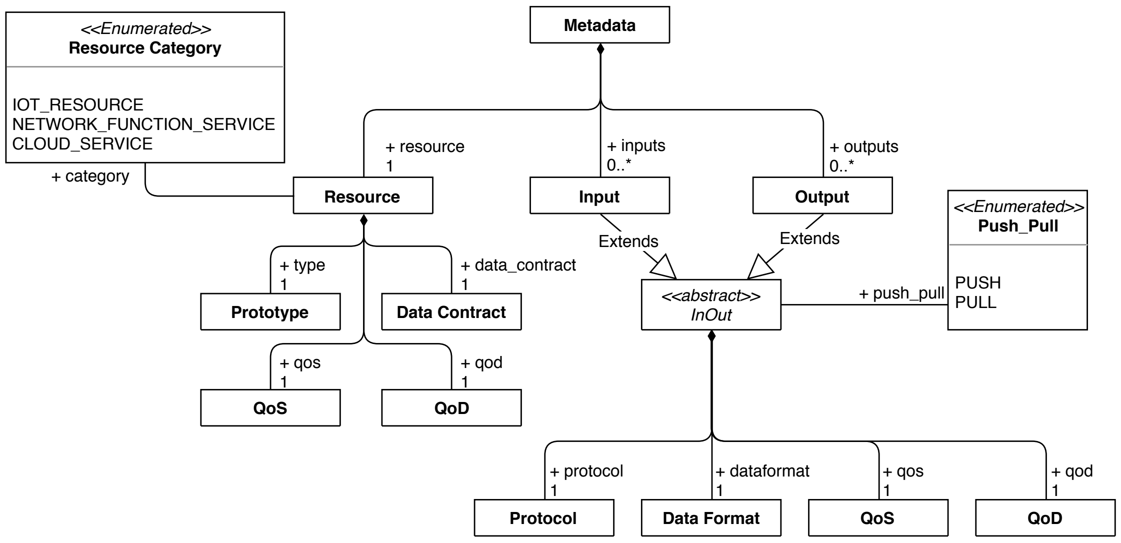


Figure 4.4: Interoperability Metadata Model

ponent can have within a graph or network topology. Inputs define the interoperability capabilities with which a SliceComponent can receive data. *Inputs* are also viewed as requirements that must be satisfied in order for the SliceComponent to work correctly. *Outputs* on the other hand describe the capabilities an SliceComponent possesses for sending data. Additionally, outputs define properties that a SliceComponent guarantees to satisfy. A comparison can be drawn to other disciplines in software engineering where preconditions must be satisfied and postconditions must be guaranteed in order for the software component to work. SliceEntities can possess multiple *Inputs* and *Outputs*, as this is often the case for actual software components.

The third main entity or category of the interoperability metadata is "*Resource*". The *Resource* category specifies capabilites and properties that are tied to the SliceComponent itself. Within the *Resoure* category, the specific type of the component can also be defined in more detail by utilising the *Prototype* class. That way, special components like message brokers or firewalls can be treated by their prototype definition and information that is relevant for the interoperability framework but specific to the type of component can also be stored within the framework.

When we defined our interoperability metadata properties, we looked at the interoperability factors that are significant for IoT cloud systems, as well as current relevant technologies. The goal is to maximise the information that is relevant for detecting and solving interoperability problems, such that resources, interoperability software artefacts and interoperability bridges with appropriate interoperability metadata values can be reused to solve problems.

To get an understanding of what is meant with each interoperability metadata property, we created an Interoperability Metadata Catalogue, where we documented and defined the semantics of the interoperability metadata properties. The current Interoperability Metadata Catalogue is considered to be an initial suggestion that is neither final nor limited, but rather intended to be extended as more knowledge has been collected.

As can be seen in Figure 4.4, the main entities (*Inputs*, *Outputs* and *Resource*) can contain instances of the following types:

- Protocols
- Dataformat
- Quality of Service
- Quality of Data
- Data Contract
- Prototype

Since Interoperability Metadata instances are represented as documents and some types can have subtypes, our framework uses type-discriminators with the key-pattern "<type>_name". The values assigned to these discriminators are the names of the subtypes.

Protocols

The attributes of these category correspond to the interoperability level 1. When we collected the interoperability metadata attributes regarding protocols, we analysed the most current specifications of the respective protocols ([CoA18], [FR18], [MQT18], [Cha18], [STO18]). The interoperability metadata that we currently defined regarding protocols can be seen in Figure 4.5.

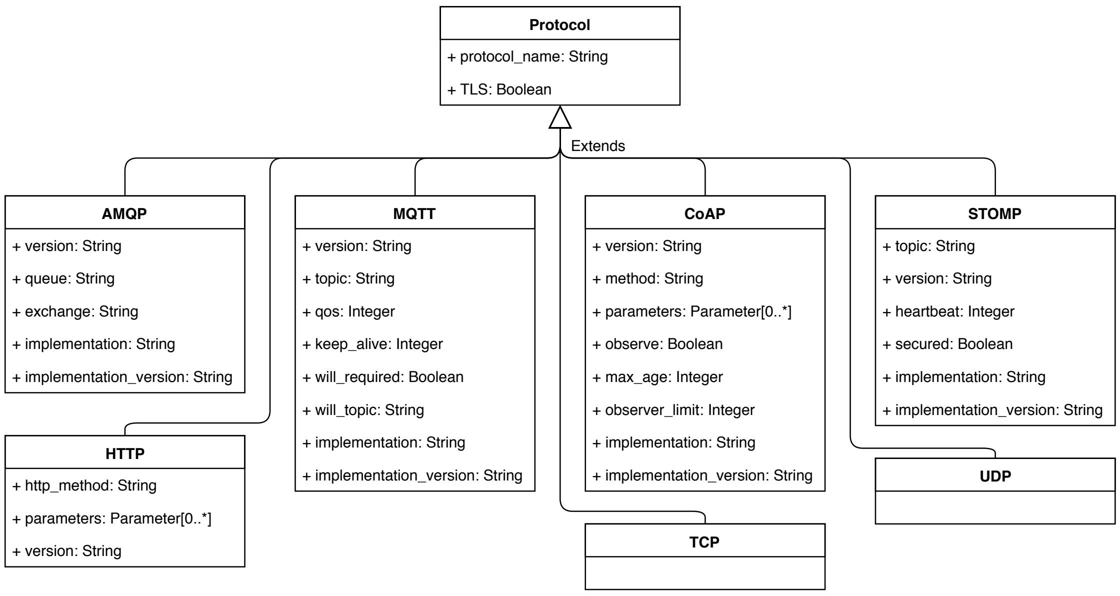


Figure 4.5: Interoperability Metadata: Protocols

Dataformats

The attributes of these category correspond to the interoperability level 1-2. Most dataformats don't require any extra attributes as the format is clearly defined. In case of csv it is important to know what seperators are used and if the header values are not added to the data stream, then the headers must also be provided in the interoperability metadata. To also handle different, well recognised data schemata (the way that the data is structured within the format), we also added the property "schema_type". The interoperability metadata that we currently defined regarding dataformats can be seen in Figure 4.6.

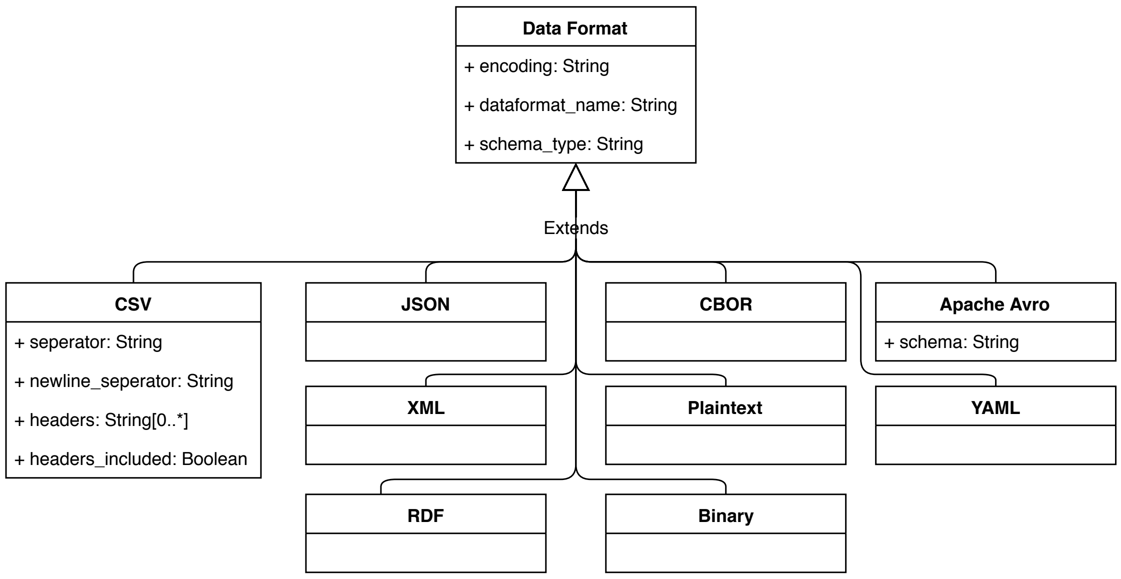


Figure 4.6: Interoperability Metadata: Dataformats

Quality of Data

The attributes of these category correspond to the additionally defined Quality of Data interoperability factor. The attributes are based on the attributes defined in [TGC+11]. The interoperability metadata that we currently defined regarding quality of data can be seen in Figure 4.7.

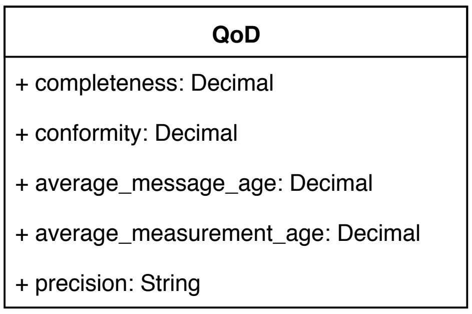


Figure 4.7: Interoperability Metadata: Quality of Data

Quality of Service

The attributes of these category correspond to the additionally defined Quality of Service interoperability factor. We defined the attributes based on conventional Quality of Service metrics. The interoperability metadata that we currently defined regarding quality of service can be seen in Figure 4.8.

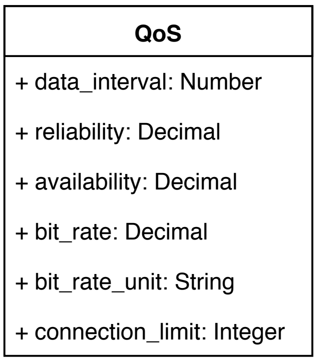


Figure 4.8: Interoperability Metadata: Quality of Service

Data Contracts

The attributes of these category correspond to the newly defined Data Contract interoperability factor. The attributes are based on the attributes defined in [TGC+11]. The interoperability metadata that we currently defined regarding data contracts can be seen in Figure 4.9.

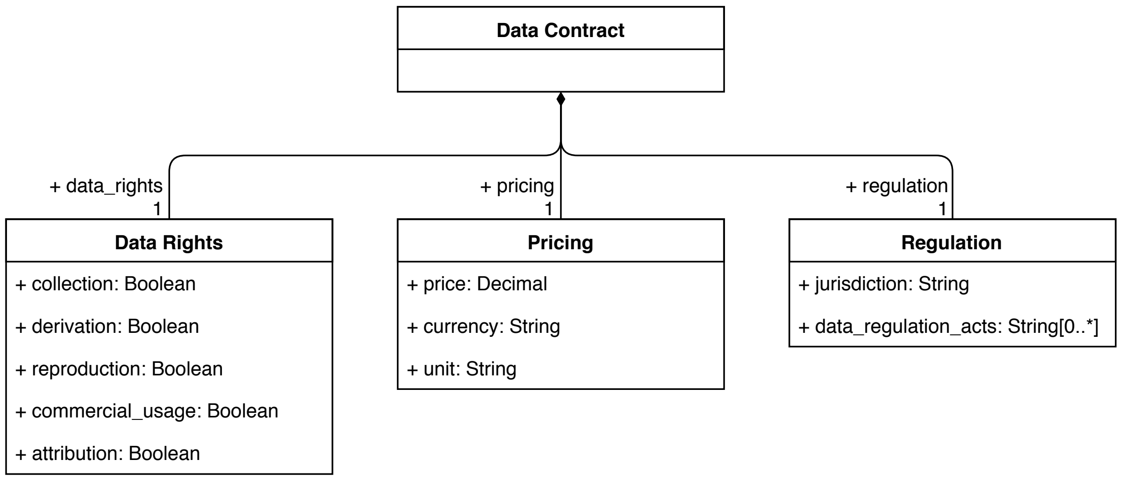


Figure 4.9: Interoperability Metadata: Data Contract

Prototypes

For this category we defined multiple prototypical components that are quite common in network topologies. As stated before, this Interoperabilty Metadata Catalogue can be extended with additional prototypes that add value to the framework. The interoperability metadata that we currently defined regarding prototypes can be seen in Figure 4.10.

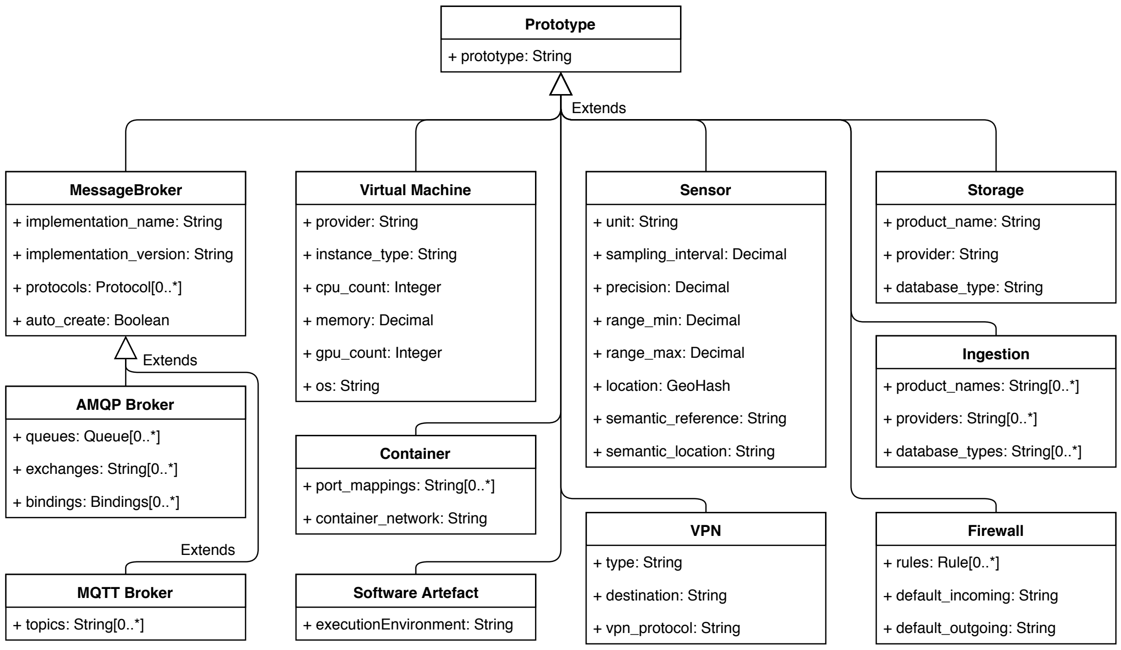


Figure 4.10: Interoperability Metadata: Prototypes

```
{
    ...
    "pcs":{
      "metadata": {
        "resource": {
          "category":"iot"
        },
        "inputs": [{
          "push_pull":"push",
          "protocol":{
            "protocol_name":"mqtt",
            "qos":2,
            "topic":"pcs_in"
          },
          "dataformat":{
            "encoding":"utf-8",
            "dataformat_name":"json"
          }
        }],
        "outputs": [{
            ...
        }]
      }
    },
    "broker": {
      "metadata": {
        "resource": {
          "category":"network_function_service",
          "type":{
            "prototype":"messagebroker",
            "protocols":[{
              "protocol_name":"mqtt"
            }],
            "topics":["pcs_in", "pcs_out"],
            "auto_create":true
          }
        },
        "inputs": [],
        "outputs": []
      },
    ...
}
```

Listing 4.2: Example of an Interoperability Metadata definition

By having information about the *Inputs* and *Outputs*, respectively requirements and guarantees, of a SliceElement, we can check if there are any interoperability problems within the SliceInformation. Additional interoperability metadata of the *Resource* category further improves the capabilities of detecting interoperability problems and also allows to check the SliceInformation against a SliceContract. Interoperability metadata can be utilised to reuse software components for interoperability purposes. The main entities that we define within our framework for supporting on-demand interoperability are interoperability software artefacts and interoperability bridges.

4.5.4 Interoperability Software Artefact

One of the interoperability framework's main entities are software artefacts, respectively interoperability software artefacts if the interoperability metadata for the software artefact is defined. Software artefacts can in general be any pieces of software or software components. The requirement our framework puts on software artefacts are the following:

- can be deployed to a capable resource (from the resource provisioning service that our framework depends on)
- can be executed by the resource that is bound to the software artefact
- can be stored and replicated (for deployment)

An example of a software artefacts could be a Node-RED flow or a python application within a Docker image. One typical use case for the an software artefact application is for instance to connect to a message broker, transform the messages it receives (for example to solve an interoperability issue regarding the message format) and forward them to another message broker. Naturally, this is just one example from an unlimited application domain. The information model of interoperability software artefacts can be seen in Figure 4.11.

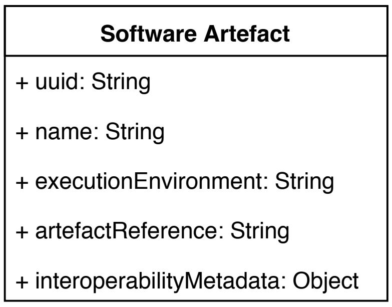

Software Artefact
+ uuid: String
+ name: String
+ executionEnvironment: String
+ artefactReference: String
+ interoperabilityMetadata: Object

Figure 4.11: Interoperability Software Artefact Model

The most important attributes of an interoperability software artefact are *executionEnvironment*, *artefactReference* and *interoperabilityMetadata*.

```
{
    "uuid": "5bbc4678e7179a6602f78ac0",
    "name": "http2datastorage",
    "executionEnvironment": "docker",
    "artefactReference": "rdsea/http2datastorage",
    "metadata": { ...
    }
}
```

Listing 4.3: Example of an Interoperability Software Artefact

executionEnvironment defines the environment in which a software artefact can be executed. This is for instance Docker for a Docker Image or Node-RED for a Node-RED flow. For those two examples (and our prototype implementation) a simple data string is sufficient to describe the domain. However, other executionEnvirionments that do not clearly define how a software component is correctly run and leave this freedom to the developer, might require additional information about how precicely to run the software artefact. With that regard, the framework can be complemented with the additionaly required information if necessary. The next attribute, *artefactReference*, determines the storage location of the software artefact. By adding *interoperabilityMetadata* to the software artefact, interoperability software artefacts can be reused. The benefit is that developers can utilise existing software artefacts for solving interoperability problems that have already been solved by someone else, and therefore spare the time it would take to develop the artefact themselves. Additionally, if the resource provisioning services provide a diverse set of resources types that are capable of providing a variety of execution environments, then software artefacts can easily be integrated to the framework. Furthermore, the software artefacts can be provisioned and executed on-demand. Listing 4.3 presents the document of a interoperability software artefact.

4.5.5 Interoperability Bridge

The second main entity for raising dynamic interoperbility with our framework are interoperability bridges. Interoperability bridges are more or less SliceInformation that can be saved and reused for interoperability purposes. The big benefit of this approach is that interoperability bridges therefore offer a powerful mechanism to quickly combine several small building blocks like software artefacts or resources to a new, more powerful component that is capable of solving more complex problems. The information model of interoperability bridges can be seen in Figure 4.12.

Interoperability bridges have their own interoperability metadata value assigment, that is condensed from the interoperability metadata of the bridge's components. Condensing and storing this information seperately decreases the time and complexity to search an interoperability bridge based on it's interoperability capabilities. Due to the direct

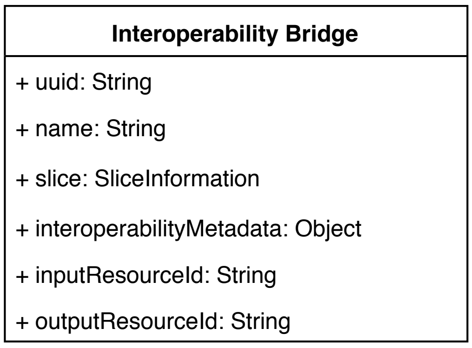

Interoperability Bridge
+ uuid: String
+ name: String
+ slice: SliceInformation
+ interoperabilityMetadata: Object
+ inputResourceId: String
+ outputResourceId: String

Figure 4.12: Interoperability Bridge Model

relation to resource slices, provisioning an interoperability bridge works the same way as provisioning a resource slice. In conclusion, interoperability bridges offer an easy but powerful mechanism to combine multiple resources and software artefacts in order to create more powerful components.

4.6 Detecting Problems - Interoperability Check

To increase the efficiency of Interoperability DevOps when analysing a SliceInformation for possible interoperability problems, the interoperability framework provides the *Interoperability Check*. The detection mechanism analyses the interoperability metadata of the SliceComponents to find problems with connected components. The first step in the detection algorithm is to determine the metadata-connections of slice-connected components.

Slice-connections versus metadata-connections

When checking the interoperability of a SliceInformation, we have to distinguish between two kinds of connections. *Slice-connections* are the connections of SliceComponents that the creator of the SliceInformation defined, or how the owner of the Slice envisions the components to interact with each other. Two SliceComponents that are slice-connected are not guaranteed to actually interoperate with each other, they are just intended to work together.

Metadata-connections on the other side are based on the interoperability metadata of two (or more) components that are meant to work together. A metadata-connection exists, if a metadata Output of the source component matches a metadata Input of the destination component. If no metadata-connection exists, then the components are meant to interoperate with each other, but they are not able to. The first step of the interoperability check algorithm is to determine all metadata-connections of the SliceInformation. However, if multiple Inputs and Outputs per component are available, this task can already be quite complex, as we also have to destinguish between direct and indirect connections.

Direct connections versus indirect connections

Even when two components are not directly connected with each other they might still interfere with each other. They are therefor indirectly connected with each other. The best example for this matter are two components (System A and B) that communicate over a message broker or have another network function like a firewall in between each other. While the communication protocol of System A and B does not have to match as long as it matches with the intermediate network function, they still have to use the same dataformat in order to interoperate with each other.

Indirect connections therefore increase the complexity of checking a SliceInformation for interoperability problems, as it is not sufficient to only check two directly connected components for errors. Another challenging task is to determine which interoperability metadata properties are relevant for an indirect connection and at what point the property is not relevant any more as it is overwritten by a more relevant component. However, this problem also arises when checking the interoperability of a SliceInformation manually. The root of the problem is that components might indirectly interoperate with each other on several layers and not every component in between might interfere with all layers.

Furthermore, having multiple Inputs and Outputs per component, as well as indirectly connected components, might lead to multiple options for chosing an Input, respectively Output. Finding the best fitted Input/Output couple is therefore not always obvious and already a complex task.

Traversing and metadata checks

Once the metadata-connections have been determined, the graph, respectively the SliceInformation can be traversed, ideally starting with SliceComponents that do not have any incoming connections. To avoid problems regarding circles, every node of the graph will be marked once it has been visited.

For every SliceComponent that is visited, the interoperability with all connected components has to be checked. We differentiate between direct and indirect connections. Based on the connection type, the approriate checks using the interoperability metadata of both components are performed. Metadata checks are usually simple comparisons using the same kind of metadata property, but can also be more complex (for instance when using specific features of the components "Prototype" category).

Checking against SliceContracts

SliceContracts contain properties that each SliceComponent has to satisfy. Checking against SliceContracts is therefore reached by simply checking the properties when visiting each component while traversing the SliceInformation.

Interoperability problem information

When an interoperability problem has been detected, the relevant information that is tied to the problem has to be reported. The information that we consider to be relevant is

- **Violated Interoperability Metadata** the keys and values of the properties that lead to the interoperability problem
- **Directly involved components** the two components between which the interoperability problem has been detected
- **Indirectly impacted components** all components that are indirectly impacted by the problem. These are all components that are on the path from source component to destination component. Additionally, if source and destination component are part of an M:N connection, then all other sources and destinations of this connection are impacted by the interoperability problem. For example, if the components S1 and S2 connect to a message broker and subsequently to the components D1 and D2, then S1 and D1 are impacted if there is an interoperability problem between S2 and D2. If there are additional components that are indirectly connected to the network function, then they are also indirectly impacted by the problem
- **Problem description** to provide understandable feedback to the Interoperability DevOp a description of the interoperability problem is highly suggested

The information about the interoperability problems of a SliceInformation can be stored in a list of problems, although a graph is deemed to be the more appropriate data structure. When recommending solutions for the interoperabilty problems, the results of the interoperability check will be used by the framework's interoperability recommendation algorithm.

4.7 Interoperability Recommendation

The interoperability framework aims to assist the Interoperability DevOps with solving interoperability problems and to reduce the time and effort necessary to solve the problems. As already described in Section 4.4, we identified three major techniques that Interoperability DevOps can apply to solve interoperability problems with our framework:

- Automatically with recommended solutions
- Manually by reusing interoperability components
- Manually by developing a new interoperability software artefact or interoperability bridge

Within all of these techniques we also identify multiple solution types that can be necessary to solve one particular interoperability problem between two components of a SliceInformation.

Solution types

We identified the following operations that can be applied to the SliceInformation to solve the interoperability problems:

- **Addition** a component will be added to the SliceInformation, typically between problem source and destination
- **Reconfiguration** the configuration of a component will be changed, if the component is reconfigurable
- **Substitution** a component will be exchanged for another one
- **Reduction** a component will be removed because it is not needed
- **Chained Addition** a chain of components will be added to the SliceInformation, typically between problem source and destination

A graphical example of each solution type can be seen in Figure 4.13. The graphs on the left-hand side represent the SliceInformation with an interoperability problem. The graphs on the right-hand side represent the SliceInformation after the problem has been solved by applying the respective solution type.

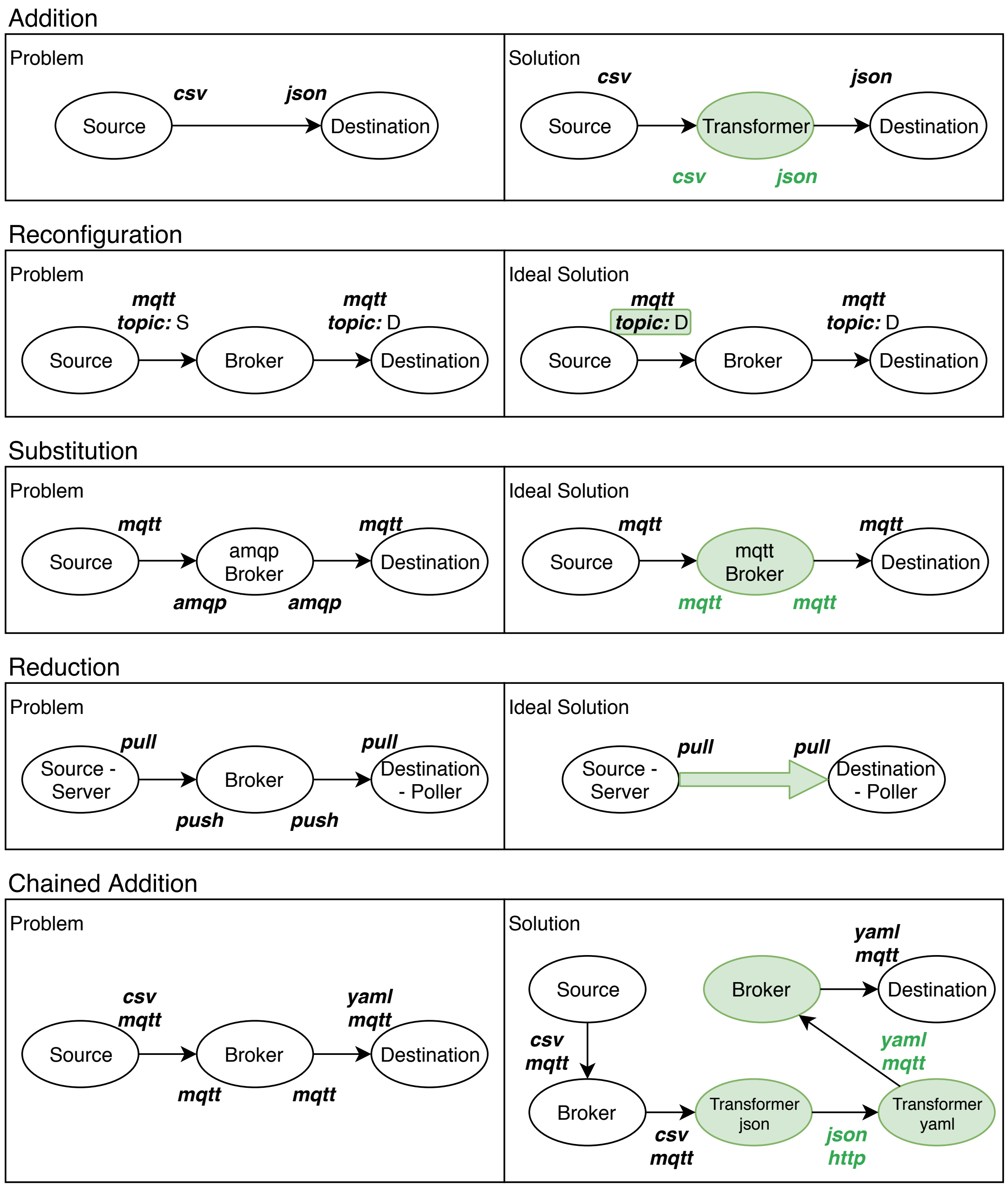


Figure 4.13: Recommendation Solution Types

We differentiate between *Addition* and *Chained Addition*, because while it is easy to find a perfect fitting component that can solve a problem (if such a component is available), finding a chain of components that solves the problem is extremely complex, with the benefit of *Chained Addition* being that a lot more problems can be solved. The more solution types an implementation of the interoperability framework is capable of applying, the higher it's ability of solving interoperability problems.

4.7.1 Automated Recommendation

An automated recommendation uses an algorithm to solve the interoperability problems of a SliceInformation. The major steps of the automated recommendation are the following:

1. **Analyse problem context** The reason for the interoperability problem as well as the components that are part of the problem have to be analysed in order to choose the appropriate solution type. If a component needs to be added to solve the problem, the position of the component is of special interest, especially if the problem is part of a 1:N, M:1 or M:N subgraph.
2. **Query for solution components** If a component needs to be added or substituted, a query has to be created that only returns components that are capable of solving the problem. The query needs to contain the interoperability metadata that caused the problem, restricted to to required values, as well as other requirements that dependent components demand to be satisfied. The new component needs to satisfy the requirements that the original component satisfied, plus the violated requirements from the interoperability metadata checks. Subsequently, SliceContract properties impose additional restrictions on the new components.
3. **Apply solution to SliceInformation** The solution must be applied to the SliceInformation. Indirectly impacted components (as described in Section 4.6) require special attention as applying the solution might have side effects on other components or other interoperability problems. In case that two interoperability problems have an overlapping impact, it is safer to re-run the interoperability check before continueing with solving the next problem.

4.7.2 Manually by reusing Interoperability Components

When solving the problem with this technique, the Interoperability DevOp has to perform all the steps that have been described for the Automated Recommendation technique her- respectively himself. For this technique, the search capabilites are of great significance, since an efficient search engine can greatly aid the Interoperability DevOps' efforts of solving the problem. The interoperability metadata serves as good foundation for searching appropriate components as it describes the components' capabilities regarding solving interoperability problems.

4.7.3 Manually by developing a new software artefact or interoperability bridge

If no components that can be reused to solve the problem are available, the final technique is to develop a new interoperability software component that is capabable of doing so.

Since developing a software application from scratch is usually a time-expensive task, the interoperability framework provides the Interoperability DevOps the possibility to create an interoperability bridge by combining already developed resources, interoperability software artefacts and other interoperability bridges.

Furthermore, if the resource provisioning service that this interoperability framework depends on, offers a diverse set of execution enviroments, then a lot of different software artefacts can be created without having to spend too much time with managing the deployment process of the software artefact, as the Interoperability Framework is capable of performing this task. Subsequently, when appropriate execution environments are provided, software artefacts can be small and simple as for instance a JavaScript function. The benefit is that small software artefacts are easier and faster to develop than a full application stack. Last but not least, the Interoperability DevOps can also acquire working software artefacts from online sources. Speeding up the development circle, respectively the time it takes to provision and deploy such software artefact greatly improves the Interoperability DevOps' ability to provide on-demand interoperability solutions. And once an interoperability software artefact has been added to the interoperability framework, it can be reused to solve a recurring problem on many occasions.

4.8 Discussion and Extensibility

4.8.1 Discussion

Interoperability metadata builds the core of the framework and is a critical building block. The semantics of metadata fields have to be well understood when assigning them. Failing to assign the correct values might result in the framework not working correctly. The catalogue of interoperability metadata properties must on the one hand be extensive, so that it is possible to make good, accurate destinctions and find as much interoperability problems as possible, which inevitably leads to complexity in using the metadata. On the other hand however, the interoperability metadata catalogue must also be clearly defined and simple to understand, so that it can be used correctly. Hence, one key factor is to find a good balance between volume (respectively accuracy) and simplicity (respectively clarity) of the interoperability metadata catalogue. Since metadata can just describe a software component and a description of a component can never be completely accurate, the goal of the framework is clearly to assist the Interoperability DevOps and not to replace them. Another key factor for a successful implementation of the framework is to focus on assisting the Interoperability DevOps as good as possible.

The Interoperability DevOps' tasks that require the most effort and time will be the tasks of finding interoperability problems and solving them. The implementation quality of these two tasks will therefore have a great impact on the implementation quality as a whole. One DevOp task that the framework does not consider right now, because of the complexity and size of the task, is the monitoring of a resource slice. By being able to easily see if a component works and access the data it produces, the efficency of creating software artefacts would be improved, as DevOps would quickly get example out- and input data. However, the size of such a monitoring framework could probably spawn another thesis. Allowing the Interoperability DevOps to extend the interoperability metadata catalogue, as well as improving the interoperablity check and recommendation capabilities is currently not part of the framework, but might be of value for an implementation of the framework.

4.8.2 Extensibility

To make the interoperability metadata dynamically extensible, following information is needed of new interoperability metadata properties:

- **Data type/domain** defines what values can be assigned to the interoperability metadata attribute
- **Identifier** denotes the name of the attribute
- **Semantic definition/description for the DevOp** a description of the attributes semantics and what qualities determine the values that need to be assigned
- **How can the interoperability metadata be merged, respectively what overides it?** This information is especially crucial if the property is used for an interoperability bridge, where components are combined to a more powerful component. Furthermore, this information has to be known when the metadata property will be used in an extended, newly defined method of the interoperability check.

In order to also use the dynamically extensible interoperability metadata for the interoperability check, or to extend the capabilities of the interoperability check as a whole, new check functions have to define the following properties:

- When does the check apply? Per component, between directly or indirectly connected components
- What is checked, respectively what criteria leads to passing respectively failing the test? The criteria may apply to the metadata of one single component, two directly or indirectly connected components, all direct or indirect neighbors of a component, all predecessors or all successors of the component. However, the check can in general be an arbitrary function as well.

The interoperability check can also be extended by calling multiple services that analyse the SliceInformation themselves. In such a case, while it would in general be sufficient to simply return whether the check failed or not, at least the information defined in Section 4.6 (Interoperability Problem Model) should be provided as feedback, in order to have enough information to comprehend and subsequently solve the problem.

4.9 Summary

Applications that need to establish interoperability with unknown entities during runtime and a constrained time window, call for on-demand interoperability solutions. On-demand interoperability is the ability to dynamically deal with interoperability problems on demand. Since a solution usually has to be found within a certain time constraint, it is important to make the tasks of dealing with dynamic interoperability problems as time-efficient as possible. As the interoperability framework is based on a resource provisioning framework, a benefitial side-effect of enabling on-demand interoperability is the increased scaleability, respectively elasticy of interoperability solutions.

As interoperability solutions have to be operated and some problems might demand a tailored interoperability solution that requires the development of a new interoperability software artefact or interoperablitiy bridge, we call for Interoperability DevOps as main stakeholders of our interoperability framework. The goal of the interoperability framework is to assist the Interoperability DevOps in their tasks and to increase their efficiency in dealing with interoperability problems, such that on-demand interoperability can be established.

Based on several interoperability factors, we defined an interoperability metadata model as the core of our framework. The interoperability metadata will be utilised to reuse software components for dealing with interoperability problems. By making use of the interoperability metadata assignments, we assist Interoperability DevOps with detecting and solving interoperability problems. The Interoperability DevOps can apply several techniques for solving the detected problems. Either with an automatically generated solution recommendation, by manually searching for capable components based on the interoperability metadata or by creating a new interoperability bridge or interoperability software artefact that is capable of solving the problem.

CHAPTER 5

Prototype

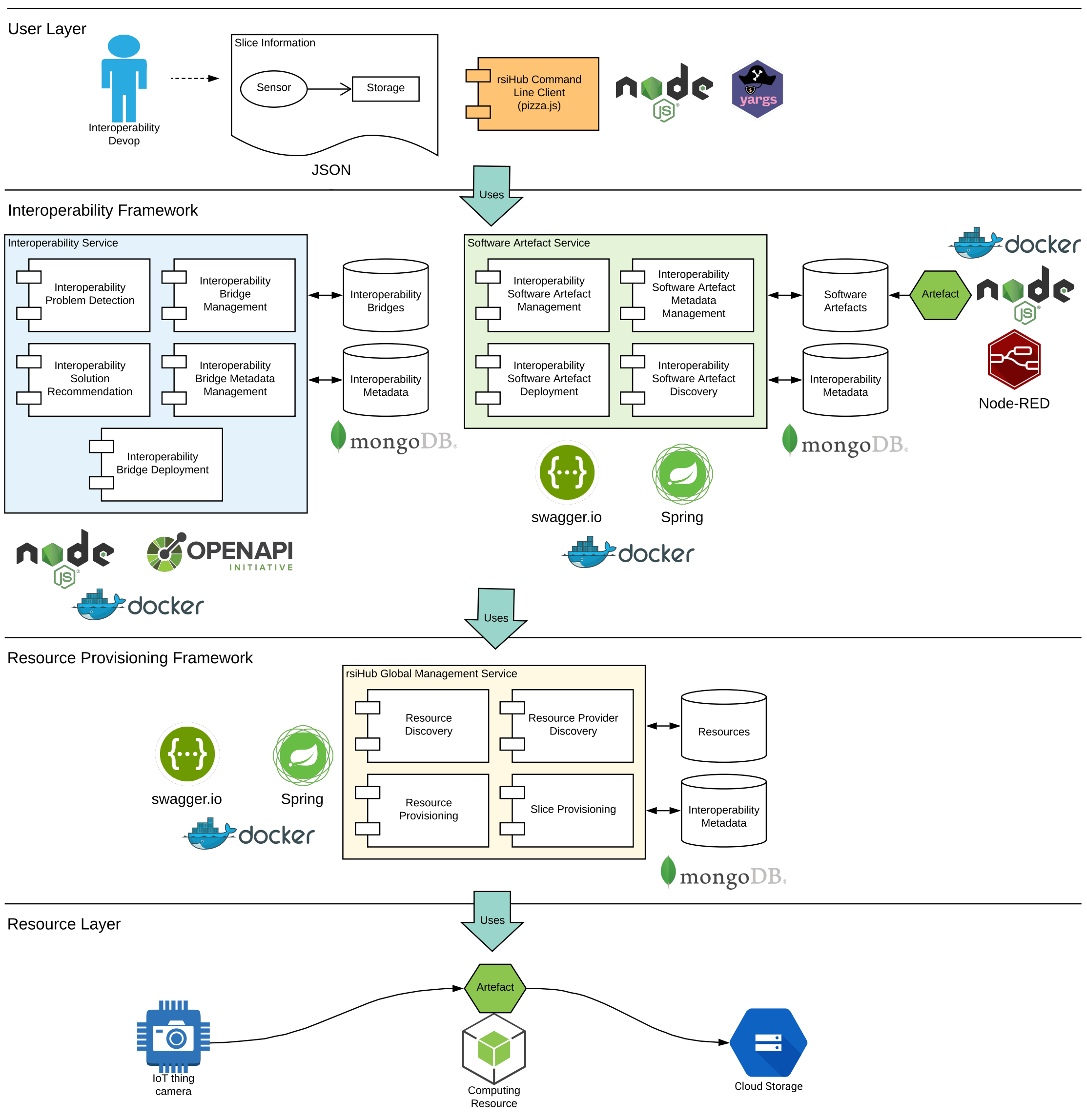
5.1 Overview

In the course of this thesis a prototype that implements the interoperability framework has been created. Since the interoperabiltiy framework requires a resource provisioning service, the prototype is built upon and part of the rsiHub Framework. Within the rsiHub framework, the main communication interface of the resource provisioning service is the Global Management Service. The prototype is available at `https://github.com/SINCConcept/HINC`

To simplify the deployment of the prototype's services, a Docker image has been created for every service within the framework and is available at Docker Hub. Additionally, as an application of the prototype is usually in the context of a specific scenario, a Scenario Generator that helps with configuring the services of a rsihub-Scenario has also been developed and is available in the repository of the prototype. Section 5.2 shows how the interoperability framework aligns with the rsiHub architecture and outlines what technologies where used for the prototype. Essential testcases that were used for the test-driven development of the prototype are introduced in Section 5.3.

5.2 Prototype Architecture and the rsiHub Framework

The frameworks that were used for creating the prototype can be seen in Figure 5.1. The



User Layer
Interoperability Devop
Slice Information
Sensor
Storage
JSON
rsiHub Command Line Client (pizza.js)
Uses
Interoperability Framework
Interoperability Service
Interoperability Problem Detection
Interoperability Bridge Management
Interoperability Solution Recommendation
Interoperability Bridge Metadata Management
Interoperability Bridge Deployment
Interoperability Bridges
Interoperability Metadata
Software Artefact Service
Interoperability Software Artefact Management
Interoperability Software Artefact Metadata Management
Interoperability Software Artefact Deployment
Interoperability Software Artefact Discovery
Software Artefacts
Artefact
Node-RED
swagger.io
Spring
Resource Provisioning Framework
rsiHub Global Management Service
Resource Discovery
Resource Provider Discovery
Resource Provisioning
Slice Provisioning
Resources
Resource Layer
IoT thing camera
Computing Resource
Cloud Storage

Figure 5.1: prototype architecture overview

Interoperability DevOp's main software component for interfacing with the interoperability framework is the *rsiHub command line client*, also called *rsiHub CLI* or *pizza.js* (since it deals with slices). *pizza.js* has been implemented with the javascript framework Node.js ([Nod18]). The Node.js package called "yargs"([Yar18]) was used in order to create a proper command line interface. The *rsiHub CLI* communicates with the rsiHub Services in order to carry our the Interoperability DevOp's tasks. All main rsiHub Services expose REST APIs that the *rsiHub CLI* uses. Interoperability DevOps therefore do not necessarily use the *rsiHub CLI*, but can also work with the REST APIs. SliceInformations are json documents that confirm to the model in Figure 4.3.

The *Interoperability Service* has also been developed with Node.js. Data is stored using MongoDB([Mon18]). The service exposes a REST API that is documented with OpenAPI Version 3([Ope18]). A Docker image of the *Interoperability Service* is available at Docker Hub with the identifier *rdsea/rsihubintop*.

The *Software Artefact Service* has been implemented in Java([Ora18]), using the Spring framework([Spr18]). MongoDB has been used for the database layer. The service also exposes a REST API that is documented with Swagger.io (OpenAPI Version 2, [Swa18]). The Docker image is available in Docker Hub with the identifier *rdsea/rsihubsas*. Some Software Artefacts that were used when developing and testing the prototype are available at `https://github.com/rdsea/IoTCloudSamples`. The artefacts have been developed with either Node-RED([NR18]) or Node.js and Docker.

As described in the previous Chapter, the Interoperability Framework depends on a resource provisioning framework. The *rsiHub Global Management Service* exposes a REST API that this prototype uses to interact with the required capabilities. The *rsiHub Global Management Service* is not part of this prototype and it was already available. During the course of this thesis, the *rsiHub Global Management Service* has merely been modified to a minor degree, in order for the interoperability framework to work.

The *rsiHub Global Management Service* depends on resource providers that are able to provision resources. The resource providers were already available and have not been modified. The resource providers were used in order to deploy software artefacts and for testing purposes in general. Specifically the resource providers that provision Node-RED and Docker instances were being used for deploying Software Artefacts.

The architecture of rsiHub containing Interoperability Software Artefact Service, Interoperability Service and the Resource Provisioning Service (rsiHub Global Management Service) together with it's utilised components, is shown in Figure 5.2.

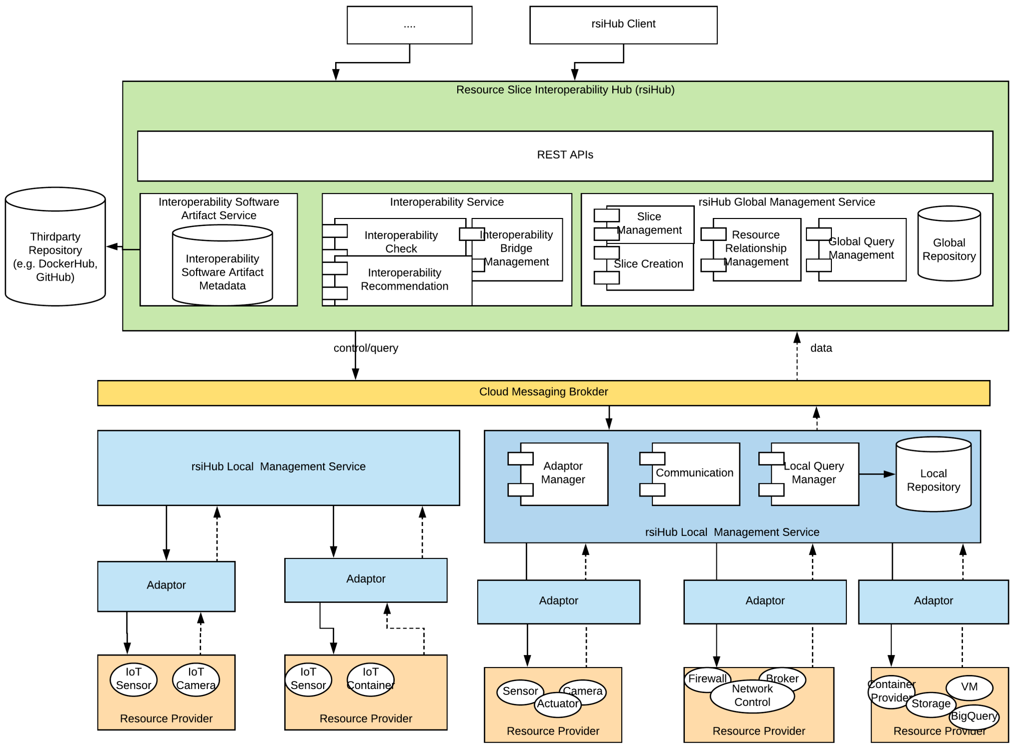


Figure 5.2: rsiHub architecture taken from [Tru18]

The prototype has been developed to satisfy the defined use cases from Section 3.5 and to allow for much flexibility in scenarios and application-domains. In the next section we present challenges that arised and testcases that we developed when we implemented the prototype in a test-driven approach.

5.3 Test-driven Development

The prototype has been developed in a test-driven approach, which turned out to be crucial in order to deal with the complexity of the interoperability check and recommendation. The time that was needed to develop the testcases turned out to provide a good return of investment. While developing the interoperability check and recommendation of the prototype, a total of 73 unit tests have been implemented, that the prototype all passes. Implemented testcases are available at `https://github.com/SINCConcept/HINC/tree/master/interoperability-service/src/test` and can be executed from the project directory with `npm test`.

Since SliceInformations can possess an arbitrary topology, it is important to find test instances that represent and cover lots of problematic cases. Apart from the SliceInformation topology, other dimensions, that further increase of problem instance domain, are the combinations of metadata fields that are available for each SliceInformation element as well as the values that are assigned to those metadata fields. For the purpose of finding test instances, the problem of having arbitrary combinations of metadata fields can be viewed from a more general but simpler perspective. Metadata values of two connected resources are always evaluated by the corresponding metadata-check. If we see the metadata-check as a function that maps two sets of metadata values (one for each resource of a connection) to the doman {true, false}, then we can project every tuple of metadata value combinations to a boolean value.

This simplification leads to the situation that, for the slices that are used as test instances, it is then not important, what metadata values are actually present, but only what the result of a comparison will be. Under this circumstance, two problem instances that differ in the metadata assignments, but are equal in the topology as well as in the connection comparison results (whether a connection is interoperable or not), can be considered equal. Nevertheless, taking this simplification into account, leads to the necessity of testing the correctness of metadata comparators seperately, as for the overall application, the correctness of metadata comparisons is vital to the application quality. Also, metadata combinations are still relevant if a problematic connection can not be solved with one perfectly fitting resource (i.e. if no resource is available that satisfies the metadata values of both the source and destination resource). For automatic solution chains, the metadata combinations are of great importance, but the calculation of such solution chains can also be tested seperately, with taking only just a few basic slice topologies into account, but greatly varying the metadata combinations and available resources.

However, the problem instances that we came up with constitute a good set for testing the prototype, as they test the very core of the detection and recommendation algorithm, as well as the other solving techniques. Tables 5.1, 5.2, 5.3 and 5.4 provide an overview and reference to the testslices that were created and used to implement the prototype. Thereafter, some essential test instances are described in more detail, to elaborate some of challenges that are embodied within the instances.

Testslices that are based on the seaport application domain

The actual slices are available at `https://github.com/SINCConcept/HINC/tree/master/interoperability-service/client_testslices/valencia_intop`

Testslice Name	**Description**	**Slice Components**	**Interoperability Problem**
`01_protocol`	This testslice corresponds to the scenario from section 3.4.1	vessel, broker, pcs	protocol mqtt to amqp (and vice versa)
`02_dataformat`	This slice corresponds to the scenario from section 3.4.2	vessel, broker, pcs	dataformat csv to json (and vice versa)
`03_datacontract_jurisdiction`	This slice corresponds to the scenario from section 3.4.3	camera, us_storage, eu_storage	camera requires eu jurisdiction
`04_datacontract_datarights`	This slice corresponds to the scenario from section 3.4.3. It must be used with the corresponding SliceContract from the subfolder `/slicecontract`.	sensor, broker, analytics	sensor requires non commercial usage
`05_datacontract_pricing`	This slice corresponds to the scenario from section 3.4.3	tracking_datasource, analytics	pricing model problem
`06_qos_reliability`	This slice corresponds to the scenario from section 3.4.4	gateaccess_sensors, broker, analytics	analytics requires reliable sensors
`07_qos_messagefrequency`	This slice corresponds to the scenario from section 3.4.4	sensor, broker, analytics	analytics requires messagefrequency
`08_qod_precision`	This slice corresponds to the scenario from section 3.4.5	sensor, broker, analytics	analytics requires precision
`09_qod_averagemeasurementage`	This slice corresponds to the scenario from section 3.4.5	tracking_datasource, analytics	analytics requires average_measurement_age

Table 5.1: Testslices that are based on the seaport application domain.

Basic testslices that represent common problem graphs (Part 1 of 2)

The actual slices are available at `https://github.com/SINCConcept/HINC/tree/master/interoperability-service/client_testslices/basic`

Testslice Name	**Description**	**Slice Components**	**Interoperability Problem**
`0_working_slice`	A working slice	httpSource, httpDest	none
`1_direct_mismatch`	A slice with a 1:1 direct connection mismatch	httpSource, httpDest	dataformat csv to json
`2_indirect_mismatch`	A slice with an 1:1 indirect connection mismatch	mqttSource, mqttBroker, mqttDest	dataformat csv to json
`3_substitution`	A slice that contains an incompatible broker that ideally should be substituted instead of adding solutions	mqttSource, amqpBroker, mqttDest	amqpBroker to mqttSource and mqttDest
`4_reduction`	A slice that contains an unnecessary broker that ideally should be removed instead of adding solutions	httpSource, mqttBroker, httpDest	mqttBroker in between httpSource and httpDest
`5_push_pull`	A slice that is similar to the previous one but can not be solved with reduction	httpSource, mqttBroker, httpDest, mqttDest	httpSource to mqttBroker, mqttBroker to httpDest. Reduction of mqttBroker not possible
`6_missing_broker`	An mqtt connection without broker	mqttSource, mqttDest	No broker is used between mqtt resources
`7_chaining`	A chain with broker and transformer should be added to this slice	mqttSource, mqttDest	missing broker + dataformat csv to json

Table 5.2: Basic testslices that represent common problem graphs (Part 1 of 2)

Basic testslices that represent common problem graphs (Part 2 of 2)

The actual slices are available at `https://github.com/SINCConcept/HINC/tree/master/interoperability-service/client_testslices/basic`

Testslice Name	**Description**	**Slice Components**	**Interoperability Problem**
`8_indirect_mismatch_m1`	A slice with an M:1 indirect mismatch	jsonSource, csvSource, broker, jsonDest	dataformat csv to json, has to be solved on source side
`9_indirect_mismatch_1n`	A slice with an 1:N indirect mismatch	jsonSource, broker, csvDest, jsonDest	dataformat csv to json, has to be solved on dest side
`10_indirect_mismatch_mn`	A slice with an M:N indirect mismatch	jsonSource, csvSource, broker, csvDest, jsonDest	multiple dataformat problems, one intermediate dataformat needs to be used (either csv or json)
`11_push_pull`	This slice represents the basic use case of the camera scenario from section 3.3	pullSource, pushDestination	http pull to google storage push
`12_basic_circle`	This slice contains a basic circle	Three components that produce and consume data, organised in a circle	two produce and consume json, one csv
`13_multi_circle`	This slice contains two direct circles and one indirect	Client, Broker, Server (Client and Server produce and consume data)	dataformat csv to json (and vice versa)

Table 5.3: Basic testslices that represent common problem graphs (Part 2 of 2)

Testslices that were derived from the slices of [GGH+15]. The slices are based on a BaseTranceiverStation scenario

The actual slices are available at `https://github.com/SINCConcept/HINC/tree/master/interoperability-service/client_testslices/bts` and `https://github.com/SINCConcept/HINC/tree/master/interoperability-service/client_testslices/lingfan_evaluation`

Testslice Name	**Description**	**Slice Components**	**Interoperability Problem**
`bts_testslice0`	A working slice of the BTS scenario	jsonSensor, Broker, Ingestion	none
`bts_testslice1`	A BTS slice with one 1:1 dataformat problem	csvSensor, Broker, Ingestion	dataformat: csv to json
`bts_testslice2`	A BTS slice with one M:1 dataformat problem	csvSensor, jsonSensor, Broker, Ingestion	dataformat: csv to json

Testslice Name	**Description**	**Slice Components**	**Interoperability Problem**
`slice_PM_tests`	A slice from the provisioning framework's prototype	Broker, Analytics, Datasink, temperatureSensor, humiditySensor	none
`slice_UC01_add_data_consumer`	A slice from the provisioning framework's prototype	Broker, Analytics, Datasink	none
`slice_UC03_protect_data_consumer`	A slice from the provisioning framework's prototype	Broker, Analytics, temperatureSensor, humiditySensor	none
`slice_UC04_custom_analysis_logic`	A slice from the provisioning framework's prototype	Broker, Analytics, Datasink, temperatureSensor, humiditySensor	none

Table 5.4: Testslices that were derived from the slices of [GGH+15]

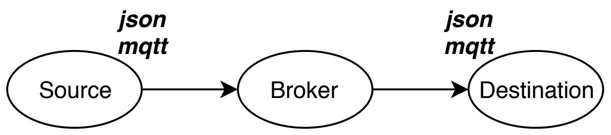
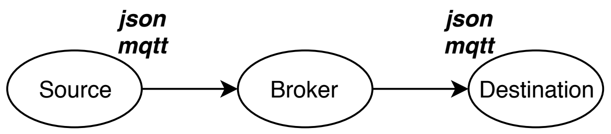
Slice correspondence	Similar to `0_working_slice` from Table 5.2
Description	A basic, working slice where the data source and destination communicate over a message broker
Problem Graph	 json mqtt json mqtt Source Broker Destination
Solution Graph	 json mqtt json mqtt Source Broker Destination
Available Resources	No additional resources are necessary for this test
Challenges	The purpose of this test is to check if the interoperability check and recommendation do not tamper the slice if it is already working.

Table 5.5: Detailed testcase description: working slice

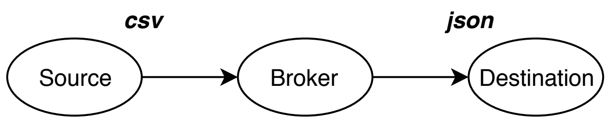
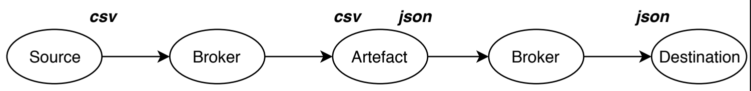
Slice correspondence	Similar to `2_indirect_mismatch` from Table 5.2
Description	A basic mismatch of an indirect connection over a message-broker
Problem Graph	
Solution Graph	 csv csv json json Source Broker Artefact Broker Destination
Available Resources	Perfect fitting solution. An artefact that translates from csv to json (see `https://github.com/rdsea/IoTCloudSamples/tree/master/IoTCloudUnits/csvToJson_nodered`. Additionally, another broker has to be provided.)
Challenges	The difficulty of this test is that the dataformat of source and destination have to be checked, while it has to be ignored for the connections to the broker. Another possible solution would be to use just one broker, but different topics.

Table 5.6: Detailed testcase description: indirect mismatch

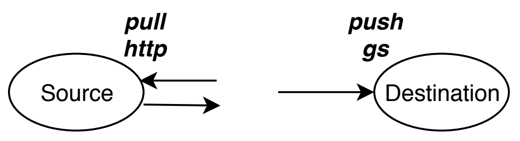
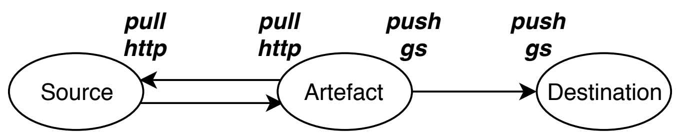
Slice correspondence	Similar to `11_push_pull` from Table 5.3
Description	A mismatch between a pulling and pushing component
Problem Graph	
Solution Graph	 pull http Source pull http Artefact push gs push gs Destination
Available Resources	Perfect fitting solution. An artefact that translates from csv to json (see `https://github.com/rdsea/IoTCloudSamples/tree/master/IoTCloudUnits/datastorageArtefact`. Additionally, another broker has to be provided.)
Challenges	This test shows the necessity of the "push/pull" field.

Table 5.7: Detailed testcase description: push pull problem

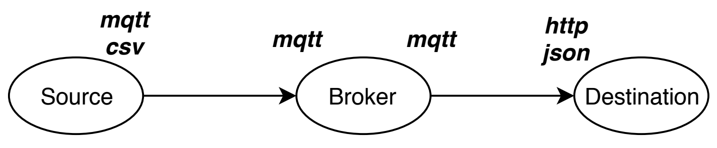
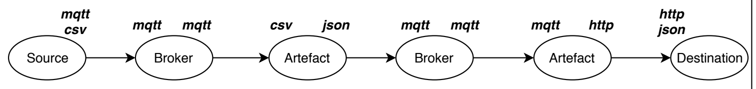
Slice correspondence	Similar to `7_chaining` from Table 5.2
Description	A problem that requires the chaining of solution components
Problem Graph	
Solution Graph	 mqtt csv Source mqtt Broker mqtt csv Artefact json mqtt Broker mqtt mqtt Artefact http http json Destination
Available Resources	Datatransformer from csv to json, Mediator from mqtt to http. An Artefact that is capable of both transformations in one step is not available for this test.
Challenges	This tests adds another difficulty, as the resources have to be chained in order to solve this problem instance.

Table 5.8: Detailed testcase description: chaining

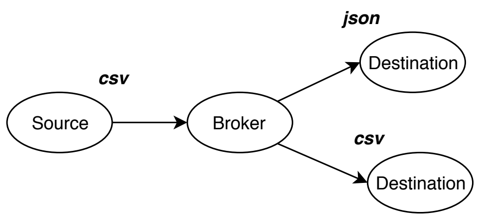
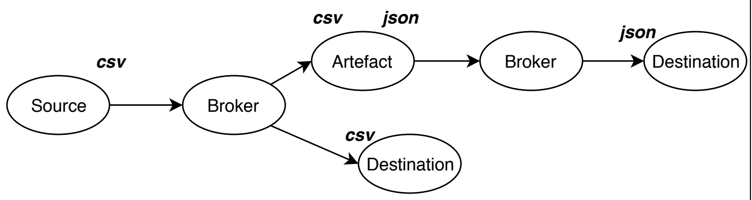
Slice correspondence	Similar to `9_indirect_mismatch_1n` from Table 5.2
Description	Indirect mismatch with one source and many destinations
Problem Graph	 json csv Source Broker Destination csv Destination
Solution Graph	 csv json csv json Source Broker Artefact Broker Destination csv Destination
Available Resources	Perfect fitting solution. An artefact that translates from csv to json (see `https://github.com/rdsea/IoTCloudSamples/tree/master/IoTCloudUnits/csvToJson_nodered`). Additionally, another broker has to be provided.
Challenges	This instance tests a 1:N connection, where only one connection from source to destination is not interoperable. The artefact therefore has to be inserted on the destination side.

Table 5.9: Detailed testcase description: indirect mismatch 1:N

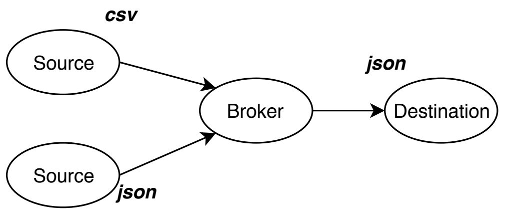
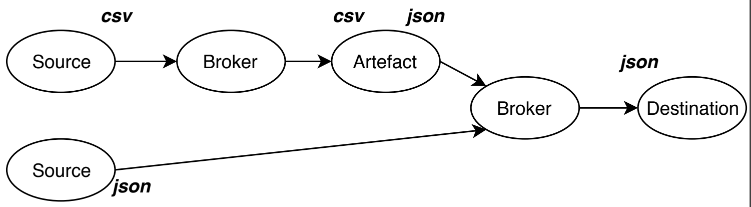
Slice correspondence	Similar to `8_indirect_mismatch_m1` from Table 5.2
Description	Indirect mismatch with many sources and one destination
Problem Graph	 csv Source json Broker Destination Source json
Solution Graph	 csv csv json Source Broker Artefact json Broker Destination Source json
Available Resources	Perfect fitting solution. An artefact that translates from csv to json (see `https://github.com/rdsea/IoTCloudSamples/tree/master/IoTCloudUnits/csvToJson_nodered`). Additionally, another broker has to be provided.
Challenges	This instance tests a M:1 connection, where only one connection from source to destination is not interoperable. The artefact therefore has to be inserted on the source side. This test, together with the previous one, shows that it is necessary to determine the side of insertion dynamically.

Table 5.10: Detailed testcase description: M:1

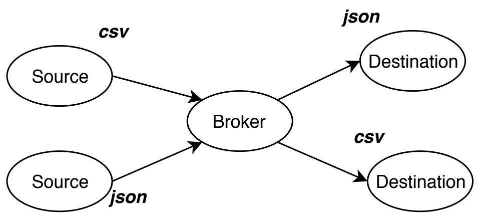
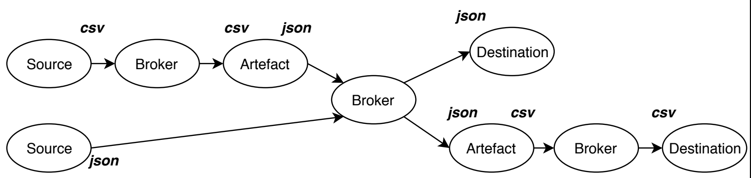
Slice correspondence	Similar to `10_indirect_mismatch_mn` from Table 5.2
Description	Indirect mismatch with many sources and many destinations
Problem Graph	 Source Source Broker Destination Destination csv json json csv
Solution Graph	 Source Broker Artefact Source Broker Destination Artefact Broker Destination csv csv json json json json csv csv
Available Resources	Perfect fitting solution. An artefact that translates from csv to json (see `https://github.com/rdsea/IoTCloudSamples/tree/master/IoTCloudUnits/csvToJson_nodered`). Additionally, another broker has to be provided.
Challenges	The difficulty of this instance is that the solutions of both problematic connections have to be coordinated. If both problems are being solved on just one side of the broker, the instance would still have two problematic connections. The symmetric solution of the one that is in the figure, would be to transform the json connections instead of the csv connections.

Table 5.11: Detailed testcase description: M:N

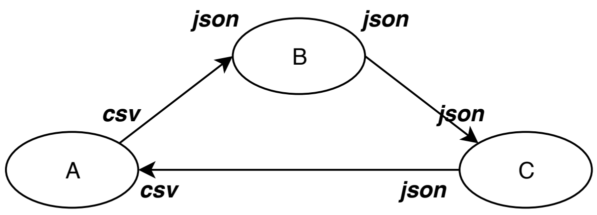
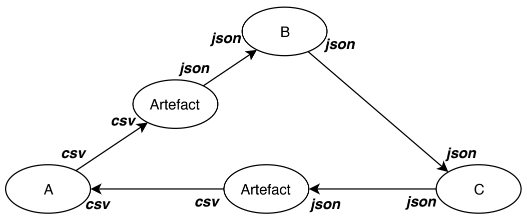
Slice correspondence	Similar to `12_basic_circle` from Table 5.2
Description	A problem within a circle
Problem Graph	
Solution Graph	 json B json json Artefact csv csv json A csv csv Artefact json json C
Available Resources	Perfect fitting solution. An artefact that translates from csv to json (see `https://github.com/rdsea/IoTCloudSamples/tree/master/IoTCloudUnits/csvToJson_nodered`). Additionally, another broker has to be provided.
Challenges	This tests checks if circles do not cause any errors.

Table 5.12: Detailed testcase description: basic circle

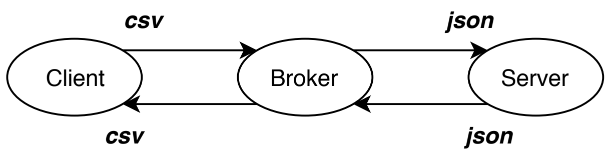
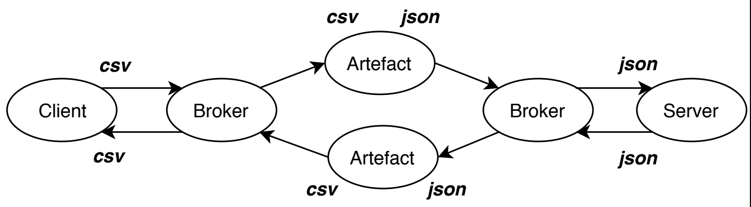
Slice correspondence	Similar to `13_multi_circle` from Table 5.2
Description	Indirect, bidirectional problem with three circles
Problem Graph	
Solution Graph	 csv json Artefact csv json Client Broker Broker Server csv Artefact json csv json
Available Resources	Perfect fitting solution. An artefact that translates from csv to json (see `https://github.com/rdsea/IoTCloudSamples/tree/master/IoTCloudUnits/csvToJson_nodered`). Additionally, another broker has to be provided.
Challenges	This test is relatively similar the detailed testcase from table 5.11 (`10_indirect_mismatch_mn`), but instead of having two Sources and Destinations, both elements (noted as "Client" and "Server") act as a source and destination simultaneously. Since they also communicate over a message broker, this test intends that they receive their own messages, so that the instance contains a total of three circles. Two small circles between message broker and "Client" respectively "Server", and one large circle that spreads over all elements. Because of the message feedbacks, both artefacts must be either on the "client" or on the "server" side of the broker, but they can not be on distributed across both broker sides.

Table 5.13: Detailed testcase description: indirect, bidirectional problem with three circles

5.4 Summary

In the course of this thesis, a prototype was created. In this chapter, we provided the architecture and frameworks that we used for building the prototype. Thereafter, we decribed the test-driven development approach that we applied for building the prototype. As the interoperability check and recommendation are complex issues, the test-driven approach was crucial. However, finding a good set of testinstance was a challenging task, due to the vast problem domain. We therefore provided a summary of our testinstance and gave detailed descriptions of several testinstance and their challenges.

CHAPTER 6

Evaluation

6.1 Overview

We evaluate our prototype along two dimension: functional and performance. At first we determine the functional appropriateness of our framework by performing the use cases from Section 3.5. We perform the use case experiments both isolated from each other and in accordance to an Interoperability DevOps workflow that we defined for the experiments. Subsequently, we evaluate our experiences against the ISO/IEC 25010:2011[ISO10] quality measures of *effectiveness*, *functional suitability* and *usability*.

Thereafter, we evaluate the performance of our prototype's interoperability check and recommendation. Our performance testinstances scale in the "amount of components per slice" and in the "amount of metadata mismatches per connection". We measure the response time to assess if the framework can reduce the effort of detecting and solving interoperability problems.

6.2 Functional Evaluation

6.2.1 Evaluation Environment

The application of the prototype is in some way always tied to a scenario. To evaluate the prototype properly, a service ecosystem is necessary, such that actual resources can be provisioned. Figure 5.2, showing the architecture of the rsiHub Framework, outlines such a service ecosystem, which consists of

- *one* rsiHub Interoperability Service
- *one* rsiHub Software Artefact Service
- *one* rsiHub Global Management Service

- *possibly several* rsiHub Local Management Services
- *possibly several* Adapter instances between rsiHub Local Management Service and Resource Provider
- *possibly several* Resource Providers
- *one* AMQP Message Broker
- *at least one* MongoDB Server

(in this listing, replicated services for the sake of load distribution only count as one)

The service ecosystem used to evaluate the prototype, is based on the application-domain from Section 3.2. As the configuration of such a service ecosystem requires quite some effort, a Scenario Generator has been developed and is available at `https://github.com/SINCConcept/HINC/tree/master/examples/src`. The Scenario Generator creates a docker-compose file and additionally the configuration files for the rsiHub services, such that the whole stack can easily be deployed to a Docker Swarm or run locally using docker-compose. This greatly reduces the effort to create a working rsiHub stack, to run it and to stop it once it is not needed anymore.
We use the stack that is created when running the scenario generator with `node src/scenarios/valencia.js` and following parameters:

- **docker swarm deployment** Yes
- **local clusters** 1
- **rabbitmq connection string** connection string of a CloudAMQP([Clo18]) instance
- **mongodb connection strings** dockerised instances

The generated rsiHub stack is then deployed to a Docker Swarm that is running in Google Cloud Platform ([Goo18a]). The Swarm consists of four VMs. The VM specification is shown in Table 6.1. After deploying the rsiHub stack to the Docker Swarm, the rsiHub

Property	Specification
Name	n1-standard-8
Number of vCPUs	8
CPU Platform	Intel Xeon E5 v4 (Broadwell E5)
Base Frequency	2.2 GHz
Memory	30 GB

Table 6.1: VM specifications for the functional evaluation

Command Line Client is configured to communicate with the rsiHub services running in the Docker Swarm, respectively Google Cloud Platform.

6.2.2 Criteria

The implementation of the proposed interoperability framework is evaluated against a selected set of quality properties from the quality in use model and the product quality model of the *ISO/IEC 25010:2011 Systems and software engineering standard* ([ISO10]). The selected set of quality properties contains the values that are deemed to be of greatest relevance for evaluating the framework, which are the following:

- **effectiveness** is defined as the *"accuracy and completeness with which users achieve specified goals"*
- **functional suitability** is defined as the *"degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions"*
- **usability** is defined as the *"degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"*

6.2.3 Experiments

For the evaluation of the functional tests we consider the most important use cases from Chapter 3. We consider the most essential use cases of this thesis and the work of Interoperability DevOps to be:

- UC01 Add Interoperability Software Artefact to rsiHub (Table 3.12)
- UC02 Deploy Interoperability Software Artefact to Resource (Table 3.13)
- UC04 Search Interoperability Software Artefact/Interoperability Bridge/Resource (Table 3.15)
- UC06 Check the interoperability of a SliceInformation (Table 3.17)
- UC08 Get interoperability recommendations for a SliceInformation (Table 3.19)

In order for the results to be reproduceable, we used a dockerised MongoDB instance that we populated with the data available at: `https://github.com/SINCConcept/HINC/blob/feature/interoperability-experiments/interoperability-service/src/test/testdata/valencia/db_resources/valencia_recommendation_evaluation.json` The steps that were carried out for testing each respective Use Case were the following:

UC01 Add Interoperability Software Artefact to rsiHub A Node-RED flow that would serve as the Software Artefact has been prepared, together with an json document that contains the Interoperability Metadata of the Artefact. The Interoperability Metadata document has been created by using the Metadata Catalogue and follows the model syntax of the Interoperability Metadata from Section 4.5.3. The MongoDB database, as well as the Cloud Storage that is used to upload Software Artefacts that are only locally available, were checked in order to guarantee that this particular Interoperability Software Artefact is not already available. By using the rsiHub CLI, the Interoperability Software Artefact has been created with `pizza artefact create <software-artefact> <execution-environment> <metadata-file> <name>`. Following parameters were used for this experiments:

- `<software-artefact>`: **`flow_json_to_csv.json`**
- `<execution-environment>`: **`nodered`**
- `<metadata-file>`: **`metadata_json_to_csv.json`**
- `<name>`: **`nodered__json__to__csv`**

After executing the command, the MongoDB database and Cloud Storage were again checked, to evaluate if the Interoperability Software Artefact has been created. Additionally, the availability of the Interoperability Software Artefact has been checked with `pizza artefact list`.

UC02 Deploy Interoperability Software Artefact to Resource This Use Case has been tested with two different approaches, either deploying the Interoperability Software Artefact to a Resource that first has to be provisioned by a Resource Provider, or alternatively to an already running Resource.

Deploy with ResourceProvider The ID of an Node-RED Interoperability Software Artefact has been retrieved. The ID of a Resource Provider that is capable of provisioning Node-RED Resources has been retrieved. We checked if no Node-RED Resource was running. By executing `pizza artefact deploy <software-artefact-id> <resource-provider-id>`, the deployment process of the Interoperability Software Artefact has been started. Following parameters were used for this experiments:

- `<software-artefact-id>`: **`5bcf22375d71560001b251ef`**
- `<resource-provider-id>`: **`nodered`**

we checked, if a new Node-RED Resource was running. By accessing the newly created Node-RED Resource, we checked if the Interoperability Software Artefact has been correctly deployed.

Deploy to running Resource The ID of an Node-RED Interoperability Software Artefact has been retrieved. The ID of a running Node-RED Resources has been retrieved. By accessing the Node-RED Resource, We checked if the Node-RED instance was empty (if the Interoperability Software Artefact was not already executed by the Resource). By executing `pizza artefact deploy <software-artefact-id> <resource-id> --useResourceID`, the deployment process of the Interoperability Software Artefact has been started. Following parameters were used for this experiments:

- `<software-artefact-id>`: **`5bcf22375d71560001b251ef`**
- `<resource-id>`: **`datatransformer1539848591979`**

By accessing the created Node-RED Resource, we checked if the Interoperability Software Artefact has been correctly deployed.

**UC04 Search Interoperability Software Artefact *
/Interoperability Bridge /Resource*** A json document that contains a MongoDB query has been prepared. The MongoDB database was checked to evaluate all Interoperability Software Artefacts that would satisfy the query. The query has been executed with `pizza artefact search <query>`. Following parameters were used for this experiments:

- `<query>`: **`'{"metadata.inputs.dataformat.dataformat_name":"json"}'`**

The returned Interoperability Software Artefacts have been checked against the query and the expected list of Interoperability Software Artefacts.

UC06 Check the Interoperability of a SliceInformation A json document that contains a SliceInformation that satsifies the model of Figure 4.3 has been prepared. The SliceInformation has been manually investigated for interoperability problems. The automated Interoperability Check has been executed with `pizza intop check <slice>`. Following parameters were used for this experiments:

- `<slice>`: **`02_dataformat.json`**

The list of interoperability problems that the prototype returned, has been evaluated against the manually registered list of interoperability problems.

UC08 Get an Interoperability Recommendation for a SliceInformation A json document that contains a SliceInformation that satsifies the model of Figure 4.3 has been prepared. The SliceInformation has been manually investigated for Introperability Problems and we checked, if the rsiHub databases contain sufficient Interoperability Software Artefacts/Resources/Interoperability Bridges to solve the interoperability problems of the SliceInformation. The automated Interoperability Recommendation has been executed with `pizza intop recommendation <slice>`. Following parameters were used for this experiments:

- `<slice>`: **`02_dataformat.json`**

The list of recommended solutions has been manually investigated and then applied to the SliceInformation. The updated SliceInformation has been checked for interoperability problems, both manually and automatically with `pizza intop check <slice>`.

6.2.4 Workflow Evaluation

In addition to the experiments that have been conducted, purely focussed on single use cases and isolated from other use cases, an experiment that is based on an evaluation workflow has also been conducted. To conduct the complete experiment, a representative workflow that is based on the Interoperability DevOps workflow diagram has been created. This evalution workflow is a fixed sequence of steps that have to be performed to deploy an interoperable slice. Since we defined several techniques to solve interoperability issues, the workflow is performed for each technique. This experiment that is based on the evaluation workflow, helps to evaluate the prototype as a whole and the interdependencies between the previously tested use cases. The evaluation workflow is shown in Figure 6.1.

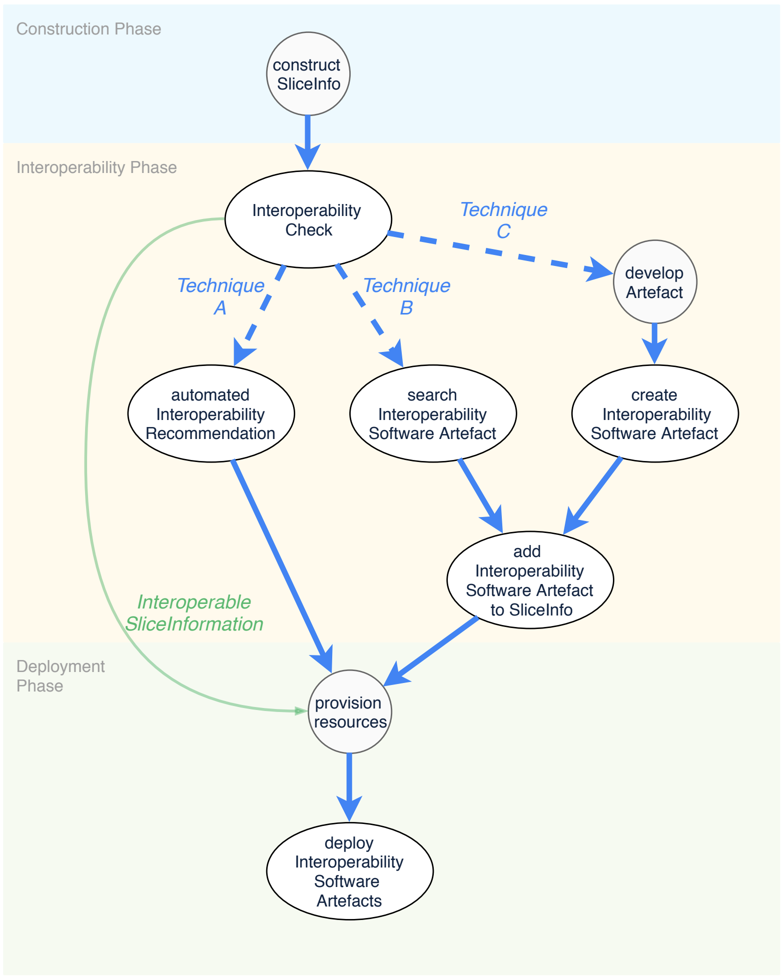

Construction Phase
construct SliceInfo
Interoperability Phase
Interoperability Check
Technique C
develop Artefact
Technique A
Technique B
automated Interoperability Recommendation
search Interoperability Software Artefact
create Interoperability Software Artefact
add Interoperability Software Artefact to SliceInfo
Interoperable SliceInformation
Deployment Phase
provision resources
deploy Interoperability Software Artefacts

Figure 6.1: Evaluation workflow

6.2.5 Evaluation Results and Discussion

Effectiveness*, the "accuracy and completeness with which users achieve specified goals"*, [ISO10].

The goal of the framework is to increase the Interoperability DevOps efficiency in dealing with interoperability problems. To achieve that, the interoperability problems need to be detected. The prototype can detect interoperability problems based on the Interoperability Metadata and solve them by reusing appropriate Software Artefacts, Resources and interoperability problems. These tasks can be done automatically, which is by far quicker than doing it manually. The efficiency goal is therefore achieved by the prototype. The "accuracy and completeness" of the effectiveness trait is also established by the prototype's capability to add new Interoperability Software Artefacts to the framework, that can then be quickly deployed. The quick deployment of newly created Interoperability Software Artefacts also leads to a quicker respond time when dealing with interoperability problems, which subsequently encourages a faster development cycle for Interoperability Software Artefacts. All those aspects speak for the Effectiveness of the prototype and framework.

On the other hand, one aspect that hinders the effectiveness of our framework is, that working with Interoperability Metadata and assigning it correctly requires expertise and can lead to errors. Especially when creating an Interoperability Software Artefact, this circumstance hinders the effectiveness of our prototype.

Functional suitability*, the "degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions"*, [ISO10].

The prototype's functions of interoperability check and interoperability recommendation as well as the management, search and deployment capabilities of Interoperability Software Artefacts and Interoperability Bridges provide a rich set of functions that can be used for properly dealing with interoperability problems.

A more appropriate way of adding Interoperability Software Artefacts, Interoperability Bridges and Resources to the SliceInformation is a feature that can improve the functionality of the prototype. Currently, this has to be done by editing the json document manually. But as this functionality is not entirely part of the Interoperability Framework alone this was not considered relevant within the scope of this thesis.

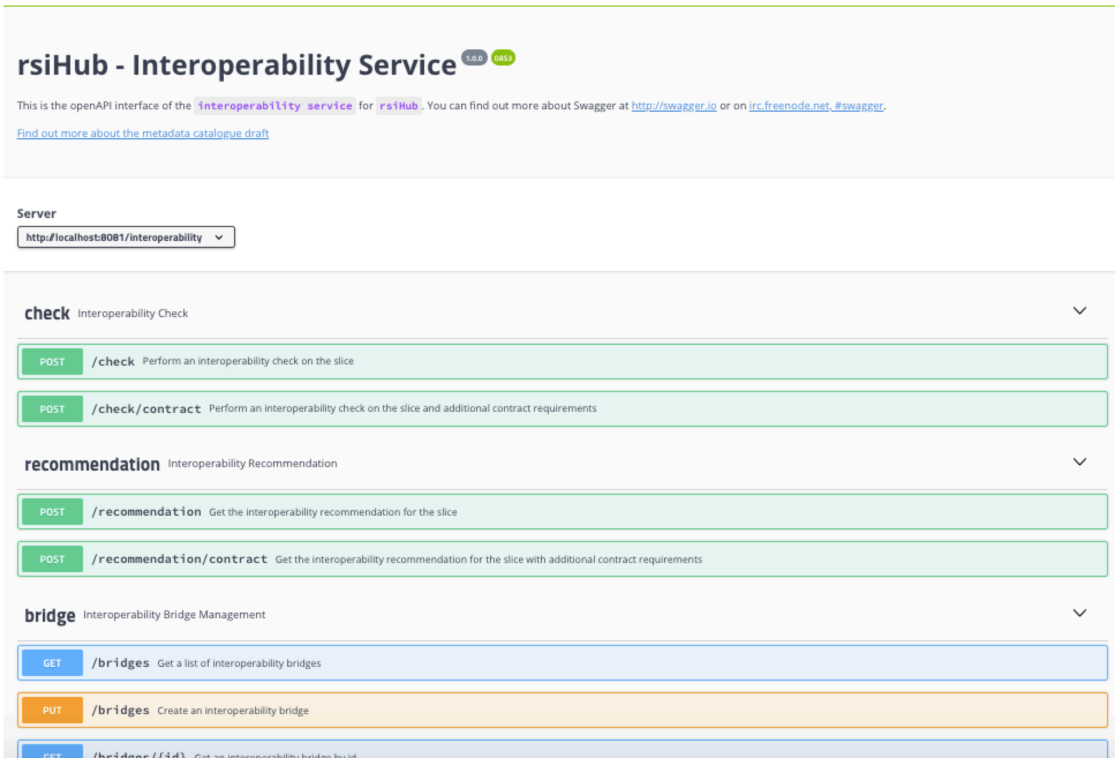
Another missing functionality, that was already mentioned in Chapter 4, is the ability to dynamically monitor problematic elements of a resource slice and access data samples in case the Interoperability Metadata is incorrect or insufficient. However, as stated before, this is a complex issue and to implement this functionality would have gone well beyond the scope of the thesis.

Usability*, the "degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"*, [ISO10].

Despite that usability was not a primary goal when developing the prototype, it is of great importance as the prototype should assist Interoperability DevOps as good as possible, hence also possess a high degree of usability. Since Interoperability DevOps are the very target group of the prototype and framework, the availability of a command line client and REST APIs, that are well-documented using OpenAPI, benefits the usability of the prototype. The rsiHub CLI can for instance be used in bash scripts, the REST API can be used by any kind of application that the Interoperability DevOps developed themselves. The rsiHub CLI's commands are also documented as expected from sophisticated command line clients and the commands are straightforward.

However, there is also room for improvement. Especially when detecting interoperability problems and recommending solutions, but also for viewing and editing a SliceInformation in general, a graphical representation of the SliceInformation would improve the usability. In such a graphical representation, the interoperability problems and recommendations could be viewed directly within the graph. The impact of a problem would then be easier and faster to comprehend.

Additionally, a feature that would simplify the manual search for interoperability solutions would be a search function that creates a query based on two Resources, Interoperability Software Artefacts or Interoperability Bridges that are not interoperable but should be connected. Currently, this is just possible by using the search capabilities that work with a MongoDB query document.



rsiHub - Interoperability Service 1.0.0 OAS3

This is the openAPI interface of the `interoperability service` for `rsiHub`. You can find out more about Swagger at http://swagger.io or on irc.freenode.net, #swagger.

Find out more about the metadata catalogue draft

Server

http://localhost:8081/interoperability

check Interoperability Check

POST /check Perform an interoperability check on the slice

POST /check/contract Perform an interoperability check on the slice and additional contract requirements

recommendation Interoperability Recommendation

POST /recommendation Get the interoperability recommendation for the slice

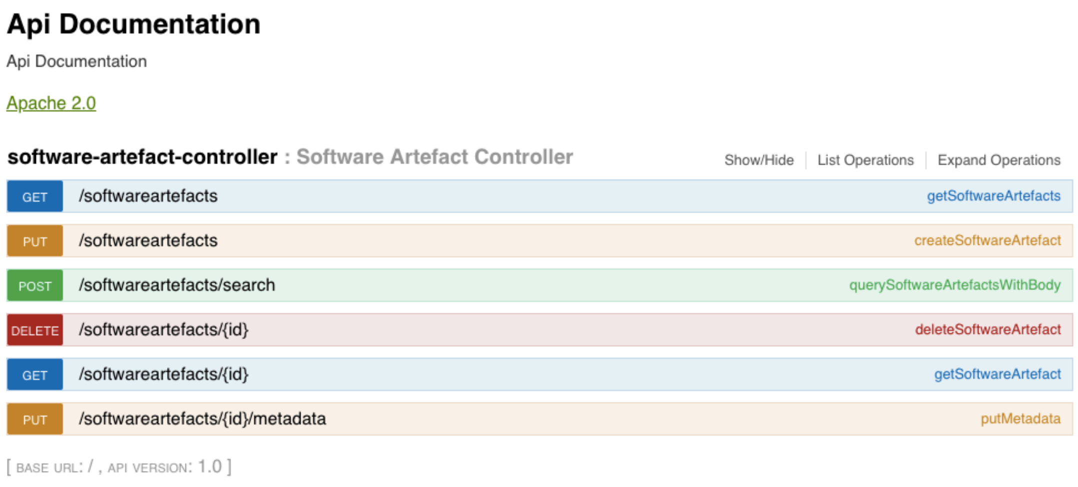
POST /recommendation/contract Get the interoperability recommendation for the slice with additional contract requirements

bridge Interoperability Bridge Management

GET /bridges Get a list of interoperability bridges

PUT /bridges Create an interoperability bridge

Figure 6.2: Screenshot of the Interoperability Service API



Api Documentation

Api Documentation

Apache 2.0

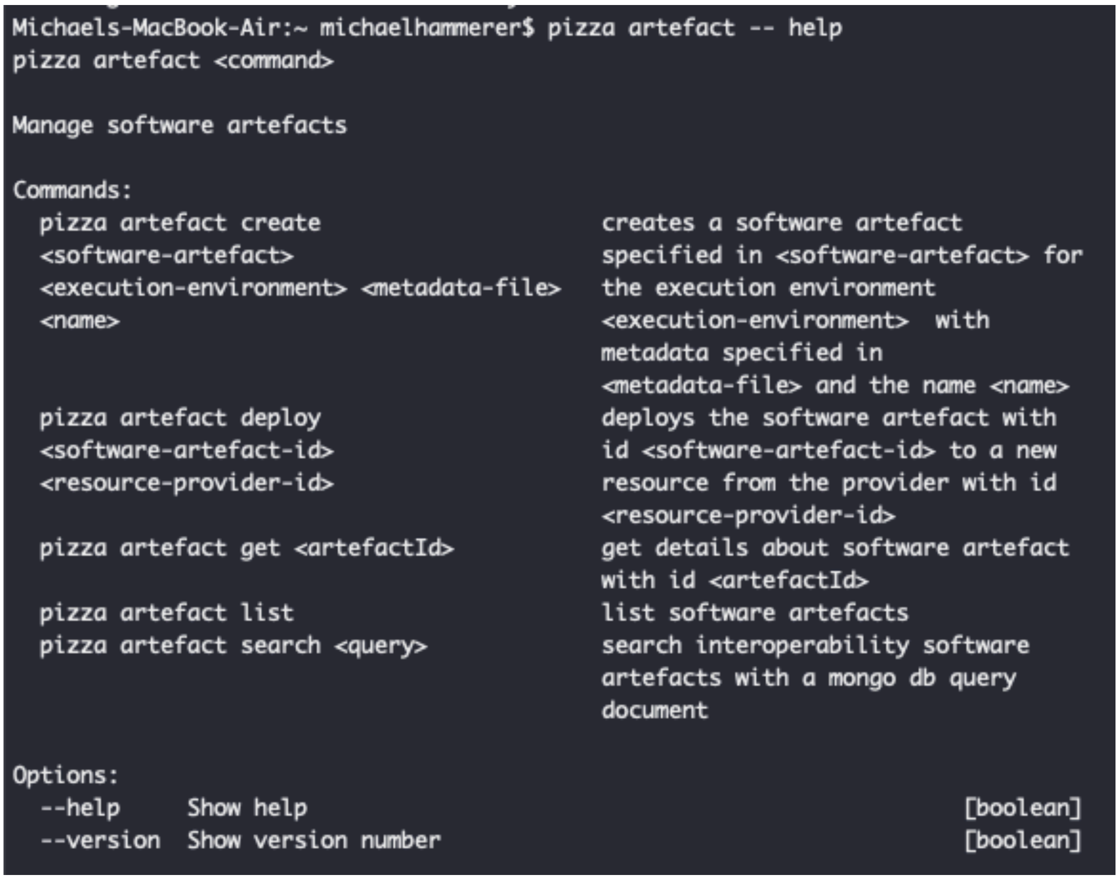
software-artefact-controller : Software Artefact Controller

Show/Hide | List Operations | Expand Operations

GET	/softwareartefacts	getSoftwareArtefacts
PUT	/softwareartefacts	createSoftwareArtefact
POST	/softwareartefacts/search	querySoftwareArtefactsWithBody
DELETE	/softwareartefacts/{id}	deleteSoftwareArtefact
GET	/softwareartefacts/{id}	getSoftwareArtefact
PUT	/softwareartefacts/{id}/metadata	putMetadata

[BASE URL: / , API VERSION: 1.0]

Figure 6.3: Screenshot of the Software Artefact Service API



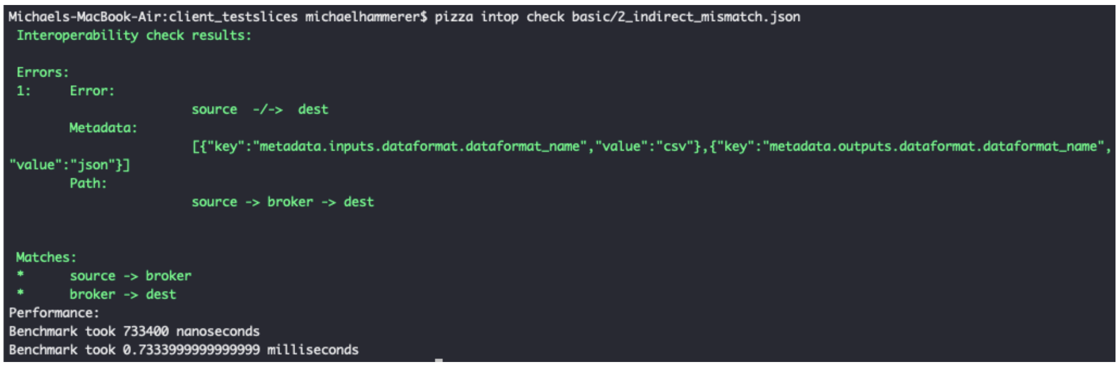
```
Michaels-MacBook-Air:~ michaelhammerer$ pizza artefact -- help
pizza artefact <command>

Manage software artefacts

Commands:
  pizza artefact create                       creates a software artefact
  <software-artefact>                         specified in <software-artefact> for
  <execution-environment> <metadata-file>     the execution environment
  <name>                                      <execution-environment>  with
                                              metadata specified in
                                              <metadata-file> and the name <name>
  pizza artefact deploy                       deploys the software artefact with
  <software-artefact-id>                      id <software-artefact-id> to a new
  <resource-provider-id>                      resource from the provider with id
                                              <resource-provider-id>
  pizza artefact get <artefactId>             get details about software artefact
                                              with id <artefactId>
  pizza artefact list                         list software artefacts
  pizza artefact search <query>               search interoperability software
                                              artefacts with a mongo db query
                                              document

Options:
  --help     Show help                                              [boolean]
  --version  Show version number                                    [boolean]
```

Figure 6.4: Software artefact commands of pizza.js

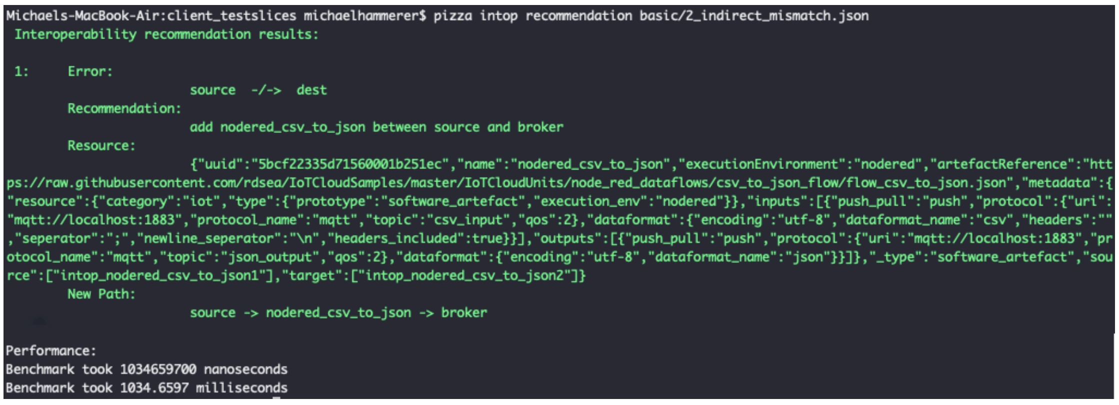


```
Michaels-MacBook-Air:client_testslices michaelhammerer$ pizza intop check basic/2_indirect_mismatch.json
 Interoperability check results:

 Errors:
 1:     Error:
                        source  -/->  dest
        Metadata:
                        [{"key":"metadata.inputs.dataformat.dataformat_name","value":"csv"},{"key":"metadata.outputs.dataformat.dataformat_name",
"value":"json"}]
        Path:
                        source -> broker -> dest


 Matches:
 *      source -> broker
 *      broker -> dest
Performance:
Benchmark took 733400 nanoseconds
Benchmark took 0.7333999999999999 milliseconds
```

Figure 6.5: Screenshot of the interoperability check with pizza.js



```
Michaels-MacBook-Air:client_testslices michaelhammerer$ pizza intop recommendation basic/2_indirect_mismatch.json
 Interoperability recommendation results:

 1:     Error:
                        source  -/->  dest
        Recommendation:
                        add nodered_csv_to_json between source and broker
        Resource:
                        {"uuid":"5bcf22335d71560001b251ec","name":"nodered_csv_to_json","executionEnvironment":"nodered","artefactReference":"htt
ps://raw.githubusercontent.com/rdsea/IoTCloudSamples/master/IoTCloudUnits/node_red_dataflows/csv_to_json_flow/flow_csv_to_json.json","metadata":{
"resource":{"category":"iot","type":{"prototype":"software_artefact","execution_env":"nodered"}},"inputs":[{"push_pull":"push","protocol":{"uri":
"mqtt://localhost:1883","protocol_name":"mqtt","topic":"csv_input","qos":2},"dataformat":{"encoding":"utf-8","dataformat_name":"csv","headers":""
,"seperator":";","newline_seperator":"\n","headers_included":true}}],"outputs":[{"push_pull":"push","protocol":{"uri":"mqtt://localhost:1883","pr
otocol_name":"mqtt","topic":"json_output","qos":2},"dataformat":{"encoding":"utf-8","dataformat_name":"json"}}]},"_type":"software_artefact","sou
rce":["intop_nodered_csv_to_json1"],"target":["intop_nodered_csv_to_json2"]}
        New Path:
                        source -> nodered_csv_to_json -> broker

Performance:
Benchmark took 1034659700 nanoseconds
Benchmark took 1034.6597 milliseconds
```

Figure 6.6: Screenshot of the interoperability recommendation with pizza.js

6.3 Performance Evaluation

6.3.1 Experiment setup

In addition to the functional evaluation, we conducted a performance test. The goal was to evaluate the response time of the interoperability check and recommendation with respect to varying instance sizes. In detail, we scaled our testinstance along two dimension. The first dimension was the number of nodes that the testinstance contained, while for the second dimension we scaled up the amount of metadata mismatches per connection.

Furthermore, besides the two scaling dimensions, we defined two different topologies that we used for the performance test. The first topology is simply a one-dimensional chain of directly-connected nodes, were each connection results in an interoperability problem, later labeled as **direct instance**. For the second topology we selected an indirect 1:M scenario, were one source would be in-directly connected to an arbitrary amount of destination-nodes (via message broker). We label the second instance type as **indirect instance**. As for the first test topology, every connection results in an interoperability problem that needs to be detected and solved. We additionally provided a mocked software artefact that would, by it's interoperability metadata values, solve the interoperability problems.

For testing the performance of our framwork, we then deployed the service stack to a Docker Swarm with one VM. For this matter, we chose a n1-standard-8 instance from Google Cloud Platform. The specifications of a n1-standard-8 instance are shown in Table 6.2

Property	**Specification**
Name	n1-standard-8
Number of vCPUs	8
CPU Platform	Intel Xeon E5 v4 (Broadwell E5)
Base Frequency	2.2 GHz
All-Core Turbo Frequency	2.8 GHz
Single-Core Max Turbo Frequency	3.7 GHz
Memory	30 GB

Table 6.2: VM specifications for the performance evaluation

6.3.2 Evaluation Results and Discussion

Figures 6.7 and 6.8 show the response time of the interoperability check, scaling the number of nodes for fixed amounts of metadata mismatches in Figure 6.7 and vice versa in Figure 6.8. The same measures are shown for the interoperability recommendation, in Figure 6.9 for the variable amount of nodes and Figure 6.10 for the variable amount of mismatches. We performed five iterations of each test, to mitigate the impact of statistical outliers. In the graphs, the datapoints of the testresults are marked with crosses. Furthermore we added a trendline that was calculated by the linear approximation. Since our performance instances are performed for up to 1000 nodes respectively metadata mismatches, we chose logarithmic scales for our graphs. While the logarithmic scales are more difficult to interpret than linear graphs, they provide more insight for our testinstances. Linear scales would render the smaller testinstances invisible in comparison to the values obtained by the larger instances.

The blue trend lines correspond to the **direct instances**, the red lines to the **indirect instances**. The lighter the trendline, the fewer is the amount of fixed metadata mismatches respectively nodes of an instance. Every graph allows for the comparison of multiple properties. We can compare the performance regarding the testinstance by looking at the differently coloured trendlines. The performance with respect to the scaling amount of nodes respectively mismatches is shown in the growth of the trendline. The influence of the fixed amount of nodes respectively metadata mismatches can be compared by looking at the colour graduation within one instance category.

We now discuss the results based on the performance graphs. The response times of the interoperability check are highly satisfactory. Even for the largest instance with 1000 slicecomponents that have 1000 metadata mismatches each, the prototype requires less than 6 seconds. Indirect checks are more costly than direct checks. At least within the performance test specification, the response time of the interoperability check grows linearily with respect to both the number of nodes and the amount of metadata mismatches.

One surprising observation that was not considered when defining the testinstances is that the interoperability recommendation performs far better for the indirect testinstances.

However, when looking at the testinstance, this result is perfectly reasonable. While the direct instance requires the solution of as many problems as the number of nodes in the instance, the indirect instance can be solved by simply adding one artefact on the source side of the 1:N relationship. For up to 100 problematic nodes where each node has 100 metadata mismatches, the interoperability recommendation provides a reasonable response time. The performance for larger instances could be improved however. As seen in the results of the indirect instance, it is important to note that the response time depends on the problematic nodes respectively the number of solutions that are required to solve the problems. This circumstance relativates the bad performance for large problem instances and deems the response times of interoperability recommendation viable for slices with a reasonable amount of problems. However, the performance can definitely be improved by applying parallel computing techniques.

Nevertheless, the automated interoperability check and recommendation provide response times that are hard to match when doing the tasks manually. Thus, our framework reduces the time to detect interoperability problems in resource slices and provides a viable tool for recommending solutions to those problems.

6.4 Summary

After describing the evaluation environment and providing the functional evaluation criteria, we presented the experiments that we conducted for the functional evaluation. The experiments are derived from use cases of Section 3.5. We conducted the use case experiments both isolated from each other and within an Interoperability DevOps workflow. Thereafter, we evaluated the *effectiveness*, *functional suitability* and *usability* of our prototype. The result of the functional evaluation is that our prototype provides an effective tool for dealing with interoperability problems. While the prototype is functionally suitable, there is room for improvement in the usability measure.

Additionally, we evaluated the performance of our prototype. We conducted the performance evaluation by measuring the prototype's response times against testinstances. We defined two different instance types that we scaled on the dimension of "mismatching metadata elements" and "nodes per slice". The performance results of the interoperability check are very satisfactory. One area where the interoperability recommendation requires improvement is when dealing with large testinstances were a lot of solutions need to be applied. However, if the resource slice contains a reasonable amount of problems, the interoperability recommendation scored satisfactory results. In conclusion, the framework can reduce the efort of dealing with interoperability problems and therefore it is considered to be a viable tool for Interoperability DevOps.

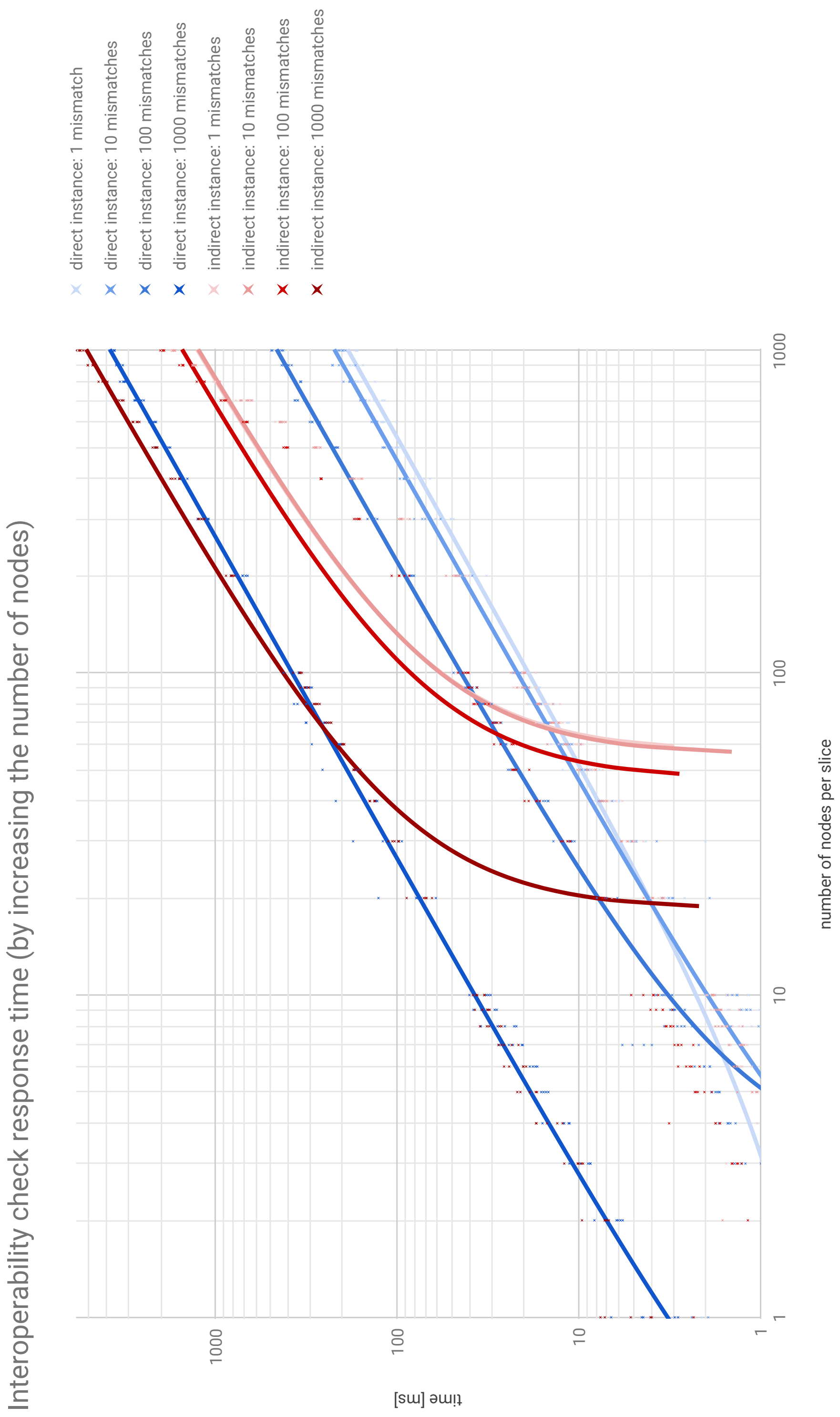


Figure 6.7: Interoperability check response time (by increasing the number of nodes)

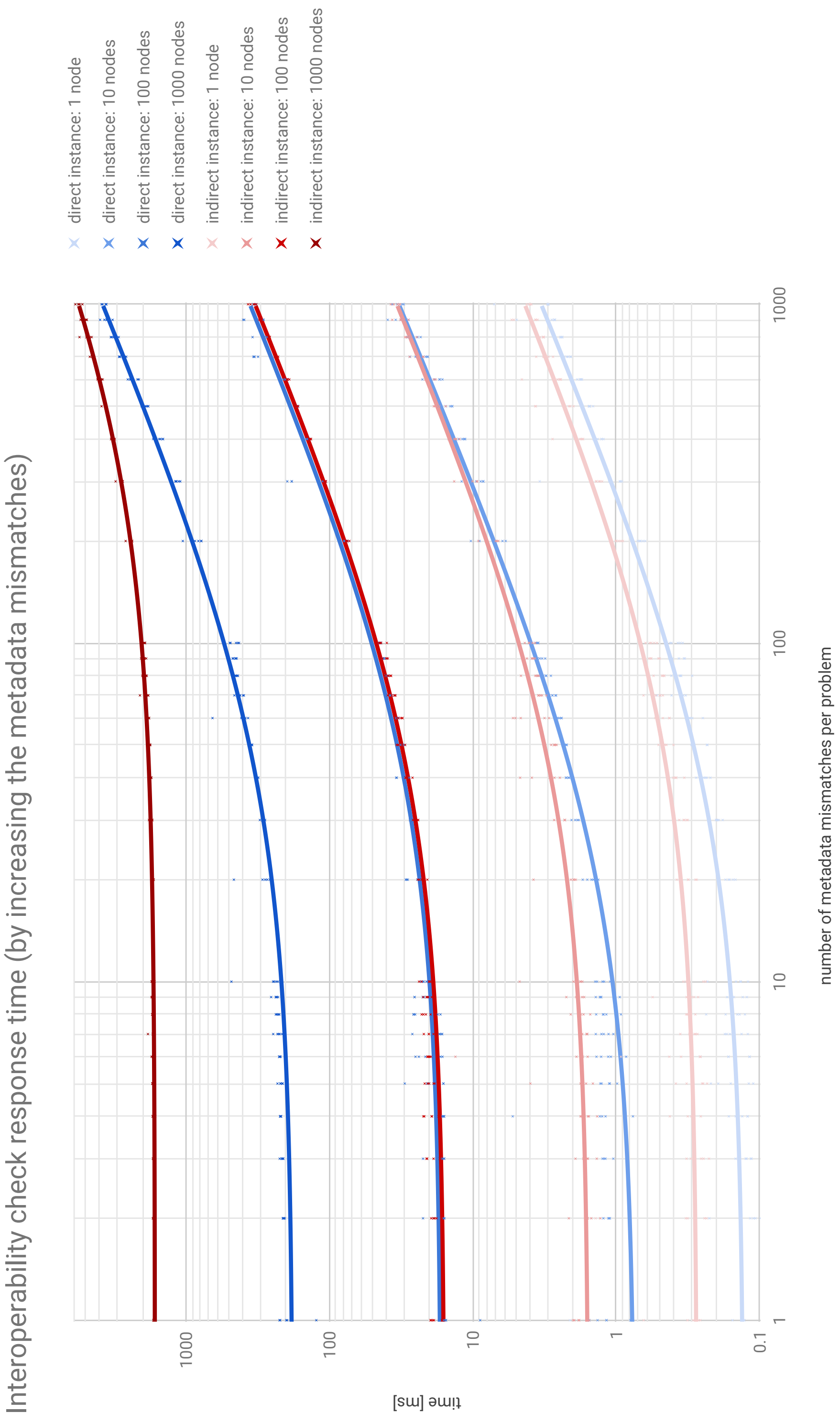


Figure 6.8: Interoperability check response time (by increasing the number of metadata mismatches)

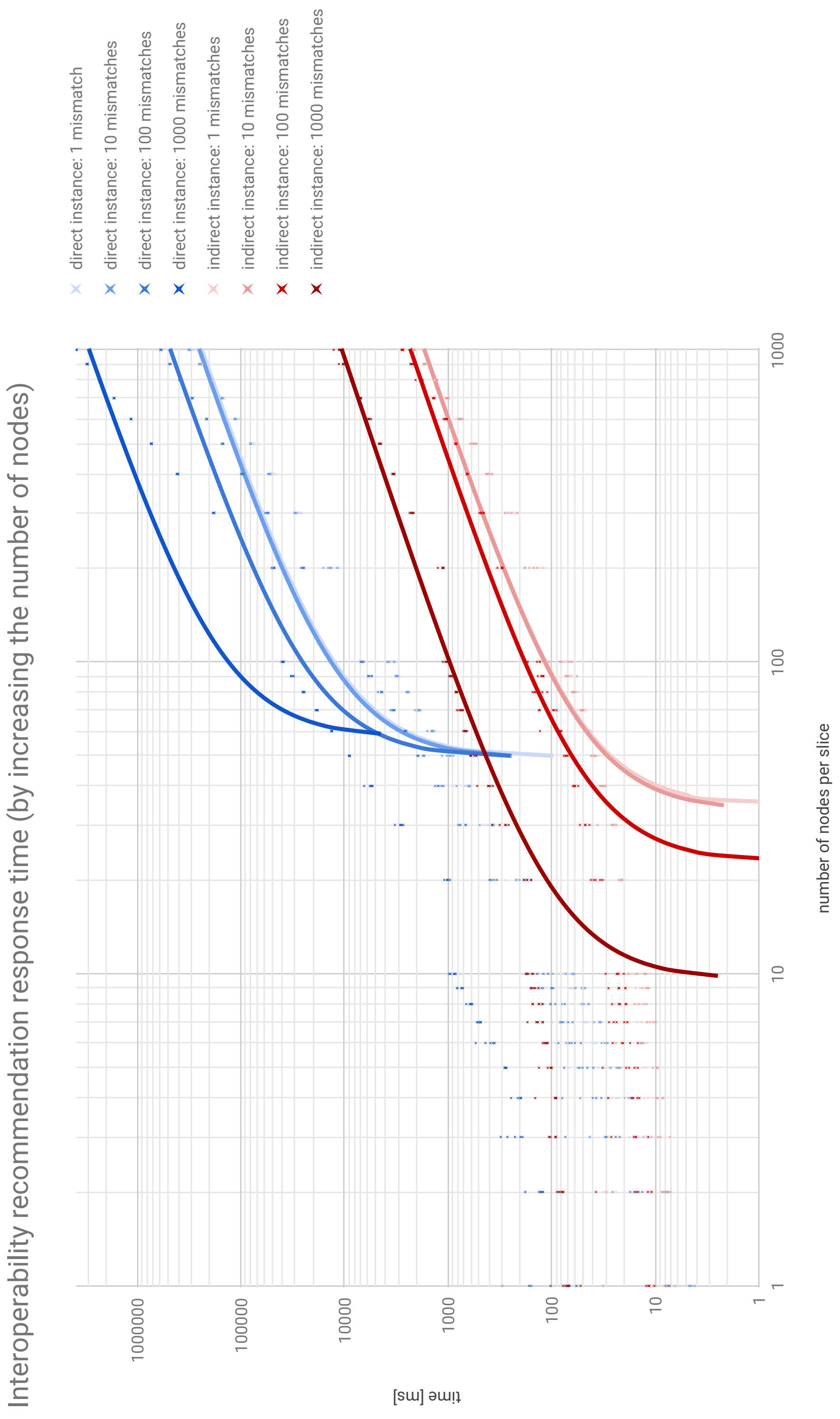


Figure 6.9: Interoperability recommendation response time (by increasing the number of nodes)

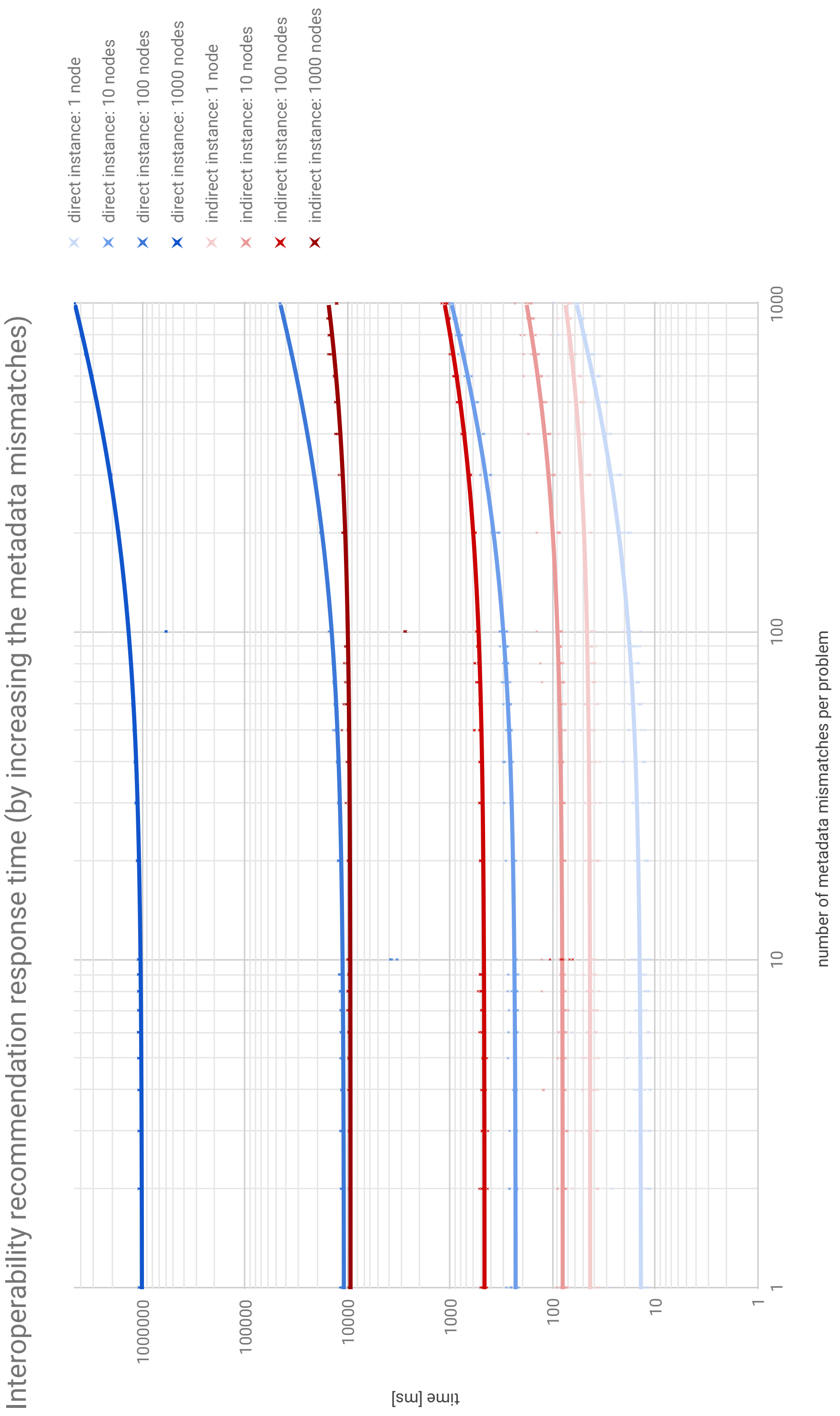


Figure 6.10: Interoperability recommendation response time (by increasing the number of metadata mismatches)

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

This thesis proposes an interoperability framework, which aims to detect and solve interoperability problems by reusing software components through interoperability metadata descriptions. The framework has been implemented and evaluated with a proof-of-concept prototype. We started the elicitiation of the framework's necessity by introducing the basic problem statement. After providing important background information, we found that currently only little to non related work exists on on-demand interoperability solutions. To get an improved understanding of the problem statement, we introduced an motivating application domain from which we derived several scenarios and subsequently use cases that were later utilised to implement the prototype.

As interoperability is a complex property that can easily be underestimated on first sight, we provided a thorough investigation on interoperability. In doing that, we started with discussing the Levels of Conceptual Interoperability Model and provided problematic interoperability examples in the scenario section. The interoperability study continued by discussing the term of on-demand interoperability and it's characteristics. Thereafter, we defined additional interoperability layers that we deem important for on-demand IoT cloud systems.

As we realised interoperability problems can not be solved entirely with the automated means of our approach, we defined and called for Interoperability DevOps to establish on-demand interoperability. We further defined the Interoperability DevOps based on their workflow and the most important tasks that it is built from. Based on the intop devop tasks, we envisioned a interoperability framework that assists the Interoperability DevOps with establishing on-demand interoperability, by decreasing the time effort that is required to perform the Interoperability DevOps' tasks.

The central point of the framework is to reuse interoperability components and resources of IoT cloud systems by describing their interoperability capabilities with interoperability metadata. The architecture of the framework, which depends on a resource provisioning service, as well as the interoperability metadata models have been described. Additionally, we defined important entities of our framework that are needed to solve interoperability problem, named interoperability software artefacts and interoperability bridges.

We utilise the interoperability metadata to analyse resource slices that represent IoT cloud systems, in order to detect interoperability problems with algorithmic measures. This functionality of our framework is called interoperability check. Based on the results of the interoperability check, we try to solve the interoperability problems by adding appropriate interoperability software artefacts, interoperability bridges and resources to the SliceInformations. This functionality is called interoperability recommendation. If the interoperability recommendation, an automated solution technique, is not able to solve the problem, the Interoperability DevOps can also solve the problem through searching for solutions manually or by creating new interoperbility software artefacts and interoperability bridges.

The framework has been implemented as a proof-of-concept prototype. The prototype is built on top of and contributes to the rsiHub framework, which offers services for provisioning IoT cloud systems based on resource slices. The prototype was built with the use cases defined in Chapter 3, in an test-driven development approach, totaling 73 automated testcases that were developed in the process. These testcases were especially valuable when dealing with the complexity of SliceInformations with respect to the interoperability check and recommendation. To simplify the configuration of the prototype and all of the resource providers that are necessary to run a specific scenario, a scenario generator has also been implemented that creates a fully configured docker stack file, since every prototype service is available as docker image.

We then evaluated the prototype's functionality against the ISO/IEC 25010:2011 quality properties of efficiency, functional suitability and usability. While the prototype offers Interoperability DevOps an efficient, foundamental set of functions, there is still room for improvement in the usability dimension. An additional functionality that would improve the prototype would be the ability to monitor components of resource slices that are not interoperable, however, this functionality would have exceeded the scope of this thesis. The performance of the prototype has also been evaluated, showing that the framework is capable of providing quick interoperability checks, even for large resource slices. When the problems can be solved with a reasonable amount of interoperability software artefacts, interoperability bridges or resources, the recommendation is also a viable tool. The response times of both interoperability check and recommendation suggest that the performance can hardly be matched with purely manual techniques. Thus, the framework reduces the time and effort required to detect and solve interoperability problems.

7.2 Future Work

As already stated above, one highly interesting feature that would add great value to the prototype, would be to provide additional data monitoring capabilities. Interoperability DevOps would then be able to easily access the data of non interoperable components. This would benefit the development process of interoperability software artefacts and in turn enhance the efficiency of our prototype. An additional feature that would increase the usability of our prototype is a graphical UI. This would provide a better overview of the SliceInformation graph and the interoperability problems that are within the SliceInformation. Furthermore, devops could recognise the context and impact of the problem faster, which would in turn facilitate solving the problem manually. Aside from improving the performance of the interoperability recommendation by thoughtfully applying parallel computing techniques, another future improvement would be to add an algorithm that is capable of recommending interoperable resource chains. Due to the complexity of such an algorithm, this feature was no within the scope of the thesis. Finally, one interesting topic would be to add a feedback loop to the interoperability check and recommendation algorithm that would provide information if a recommendation was actually successful or not. This information could be used to create a statistical model for the interoperability metadata, check and recommendation. Subsequently, the quality of checks, recommendations and the interoperability metadata catalogue could be improved. Nevertheless, such a feedback loop requires a broad application of the framework in order to gather valuable feedback information.

Bibliography

[AOK$^+$15] Muhammad Intizar Ali, Naomi Ono, Mahedi Kaysar, Keith Griffin, and Alessandra Mileo. A Semantic Processing Framework for IoT-Enabled Communication Systems. In *The Semantic Web - ISWC 2015*, Lecture Notes in Computer Science, pages 241–258. Springer, Cham, October 2015.

[BFB$^+$11] Prith Banerjee, Richard Friedrich, Cullen Bash, Patrick Goldsack, Bernardo A. Huberman, John Manley, Chandrakant Patel, Parthasarathy Ranganathan, and Alistair Veitch. Everything as a Service: Powering the New Information Economy. *Computer*, 44(3):36–43, 2011.

[BT17] F. B. Balint and H. L. Truong. On Supporting Contract-Aware IoT Dataspace Services. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 117–124, April 2017.

[Cha18] JPMorgan Chase. AMQP specification. page 113, 2018. *docs.oasis-open.org*, 2018. [Online]. Available: http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf. [Accessed: 10- Dec- 2018].

[Clo18] CloudAMQP. CloudAMQP - RabbitMQ as a Service, 2018. *cloudamqp.com*, 2018. [Online]. Available: https://www.cloudamqp.com/. [Accessed: 10- Dec- 2018].

[CoA18] CoAP. The Constrained Application Protocol (CoAP), 2018. *tools.ietf.org*, 2018. [Online]. Available: https://tools.ietf.org/html/rfc7252 . [Accessed: 10- Dec- 2018].

[DED17] H. Derhamy, J. Eliasson, and J. Delsing. IoT Interoperability #x2014;On-Demand and Low Latency Transparent Multiprotocol Translator. *IEEE Internet of Things Journal*, 4(5):1754–1763, October 2017.

[DFZ$^+$15] Yucong Duan, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C. Narendra, and Bo Hu. Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 621–628. IEEE, 2015.

[FR18] Roy Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1), 2018. *tools.ietf.org*, 2018. [Online]. Available: https://tools.ietf.org/html/rfc7230 . [Accessed: 10- Dec- 2018].

[FSP+18] Giancarlo Fortino, Claudio Savaglio, Carlos E. Palau, Jara Suarez de Puga, Maria Ganzha, Marcin Paprzycki, Miguel Montesinos, Antonio Liotta, and Miguel Llop. Towards Multi-layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach. In Raffaele Gravina, Carlos E. Palau, Marco Manso, Antonio Liotta, and Giancarlo Fortino, editors, *Integration, Interconnection, and Interoperability of IoT Systems*, Internet of Things, pages 199–232. Springer International Publishing, Cham, 2018.

[Gao18] Lingfan Gao. *On Provisioning and Configuring Ensembles of IoT, Network Functions and Cloud Resources.* Wien, Wien, 2018.

[GGH+15] V. Gazis, M. Görtz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, F. Zeiger, and E. Vasilomanolakis. A survey of technologies for the internet of things. In *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1090–1095, August 2015.

[Goo18a] Google. Google Cloud platform, 2018. *cloud.google.com*, 2018. [Online]. Available: https://cloud.google.com/. [Accessed: 10- Dec- 2018].

[Goo18b] Google. Google Sheets: Free Online Spreadsheets for Personal Use, 2018. *google.com*, 2018. [Online]. Available: https://www.google.com/sheets/about/. [Accessed: 12- Dec- 2018].

[GPS16] Paul Grace, Brian Pickering, and Mike Surridge. Model-driven interoperability: engineering heterogeneous IoT systems. *Annals of Telecommunications*, 71(3-4):141–150, April 2016.

[GS16] Amelie Gyrard and Martin Serrano. Connected Smart Cities: Interoperability with SEG 3.0 for the Internet of Things. pages 796–802. IEEE, March 2016.

[II18a] INTER-IoT. INTER-IoT - Interoperability Internet of Things, 2018. *inter-iot-project.eu*, 2018. [Online]. Available: https://http://www.inter-iot-project.eu/. [Accessed: 04- Dec- 2018].

[II18b] INTER-IoT. Inter-iot eu project open calls, 2018. *inter-iot-project.eu*, 2018. [Online]. Available: http://www.inter-iot-project.eu/open-call . [Accessed: 04- Dec- 2018].

[IoT18a] IoTivity. Architecture Overview | IoTivity, 2018. *iotivity.org*, 2018. [Online]. Available: https://iotivity.org/documentation/architecture-overview. [Accessed: 04- Dec- 2018].

[IoT18b] IoTivity. Iotivity | an open source software framework enabling seamless device-to-device connectivity to address the emerging needs of the internet of things., 2018. *iotivity.org*, 2018. [Online]. Available: https://iotivity.org/ . [Accessed: 04- Dec- 2018].

[ISO10] ISO/IEC. Iso/iec 25010 system and software quality models. Technical report, 2010.

[JAK$^+$16] Michael Jacoby, Aleksandar Antonić, Karl Kreiner, Roman Łapacz, and Jasmin Pielorz. Semantic Interoperability as Key to IoT Platform Federation. In *Interoperability and Open-Source Solutions for the Internet of Things*, Lecture Notes in Computer Science, pages 3–19. Springer, Cham, November 2016.

[LNT16] Duc-Hung Le, Nanjangud Narendra, and Hong-Linh Truong. HINC-harmonizing diverse resource information across iot, network functions, and clouds. In *Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th International Conference on*, pages 317–324. IEEE, 2016.

[LW14] Gang Li and Mingchuan Wei. Everything-as-a-service platform for on-demand virtual enterprises. *Information Systems Frontiers*, 16(3):435–452, 2014.

[MCB$^+$15] James Manyika, Michael Chui, Peter Bisson, Jonathan Woetzel, Richard Dobbs, Jacques Bughin, and Dan Aharon. Unlocking the potential of the Internet of Things | McKinsey & Company, 2015.

[Mil15] Milan Milenkovic. A case for interoperable iot sensor data and meta-data formats: The internet of things (ubiquity symposium). *Ubiquity*, 2015(November):2, 2015.

[Mon18] MongoDB. Mongodb | open Source Document Database, 2018. *mongodb.com*, 2018. [Online]. Available: https://www.mongodb.com/index . [Accessed: 10- Dec- 2018].

[MQT18] MQTT. MQTT Version 3.1.1, 2018. *docs.oasis-open.org*, 2018. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html . [Accessed: 10- Dec- 2018].

[Nod18] Node.js. Node.js | a javascript runtime built on chrome's v8 javascript engine., 2018. *nodejs.org*, 2018. [Online]. Available: https://nodejs.org/en/. [Accessed: 10- Dec- 2018].

[NR18] Node-RED. Node-RED, 2018. *nodered.org*, 2018. [Online]. Available: https://nodered.org/. [Accessed: 10- Dec- 2018].

[OCF18] OCF. OCF - oneIoTa Data Model Tool, 2018. *openconnectivity.org*, 2018. [Online]. Available: https://openconnectivity.org/developer/oneiota-data-model-tool . [Accessed: 11- Dec- 2018].

[ODO18] ODO. interoperability | Definition of interoperability in English by Oxford Dictionaries, 2018. [Online]. Available: https://en.oxforddictionaries.com/definition/interoperability . [Accessed: 04- Dec- 2018].

[Ope18] OpenAPI. Openapi | openapi initiative, 2018. *openapis.org*, 2018. [Online]. Available: https://www.openapis.org/ . [Accessed: 10- Dec- 2018].

[Ora18] Oracle. Java Software | Oracle, 2018. *oracle.com*, 2018. [Online]. Available: https://www.oracle.com/java/. [Accessed: 11- Dec- 2018].

[Pos18] Pos. IoT Standards & Protocols Guide | 2018 Comparisons on Network, Wireless Comms, Security, Industrial, 2018. *Postscapes.com*, 2018. [Online]. Available: https://www.postscapes.com/internet-of-things-protocols/. [Accessed: 04- Dec- 2018].

[SBK+17] Stefan Schmid, Arne Bröring, Denis Kramer, Sebastian Käbisch, Achille Zappa, Martin Lorenz, Yong Wang, Andreas Rausch, and Luca Gioppo. An Architecture for Interoperable IoT Ecosystems. In Ivana Podnar Žarko, Arne Broering, Sergios Soursos, and Martin Serrano, editors, *Interoperability and Open-Source Solutions for the Internet of Things*, volume 10218, pages 39–55. Springer International Publishing, Cham, 2017. DOI: 10.1007/978-3-319-56877-5_3.

[Spr18] Spring. spring.io | spring: the source for modern java, 2018. *spring.io*, 2018. [Online]. Available: https://spring.io/. [Accessed: 10- Dec- 2018].

[STO18] STOMP. Stomp specification, 2018. *stomp.github.io*, 2018. [Online]. Available: http://stomp.github.io/stomp-specification-1.2.html . [Accessed: 10- Dec- 2018].

[Swa18] Swagger.io. OpenAPI Specification | Swagger, 2018. *swagger.io*, 2018. [Online]. Available: https://swagger.io/specification/. [Accessed: 10- Dec- 2018].

[TD15] H. L. Truong and S. Dustdar. Principles for Engineering IoT Cloud Systems. *IEEE Cloud Computing*, 2(2):68–76, March 2015.

[TGC+11] H. L. Truong, G. R. Gangadharan, M. Comerio, S. Dustdar, and F. De Paoli. On Analyzing and Developing Data Contracts in Cloud-Based Data Marketplaces. In *2011 IEEE Asia-Pacific Services Computing Conference*, pages 174–181, December 2011.

[TGH18] Hong-Linh Truong, Lingfan Gao, and Michael Hammerer. *Service Architectures and Dynamic Solutions for Interoperability of IoT, Network Functions and Cloud Resources.* July 2018.

[TN16] Hong-Linh Truong and Nanjangud Narendra. SINC-An Information-Centric Approach for End-to-End IoT Cloud Resource Provisioning. In *Cloud Computing Research and Innovations (ICCCRI), 2016 International Conference on*, pages 17–24. IEEE, 2016.

[Tru18] Hong-Linh Truong. Towards a Resource Slice Interoperability Hub for IoT. In *3rd edition of Globe-IoT 2018: Towards Global Interoperability among IoT Systems, IEEE, 2018. Orlando, Florida, USA, April 17-20, 2018.*, 2018. To appear.

[WTW09] Wenguang Wang, Andreas Tolk, and Weiping Wang. The Levels of Conceptual Interoperability Model: Applying Systems Engineering Principles to M&S. In *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, pages 168:1–168:9, San Diego, CA, USA, 2009. Society for Computer Simulation International.

[Yar18] Yargs. yargs | yargs the modern, pirate-themed, successor to optimist., 2018. *npmjs.com*, 2018. [Online]. Available: https://www.npmjs.com/package/yargs . [Accessed: 10- Dec- 2018].