Universität Augsburg
Fakultät für Angewandte
Informatik

Research and Development of Interoperability Concepts for IoT Platforms

Inaugural dissertation presented for the degree of
Doctor rerum naturalium (Dr. rer. nat.) at the
Faculty of Applied Computer Science of the
University of Augsburg

by Denis Kramer

2021

Primary reviewer:	Professor Dr. Jörg Hähner
Secondary reviewer:	Professor Dr. Bernhard Bauer
Date of oral examination:	21. June 2021

Abstract

The Internet of Things (IoT) has transformed already many areas of our modern life. IoT devices are being connected through IoT platforms and new applications & services are created based on IoT data to provide added value for customers. Interoperability between different IoT platforms is still one of the most important gaps inside the Internet of Things. Current interoperability solutions are still largely driven by static & inflexible interoperability mechanisms originating from the past webservice-era which do not fit the characteristics of modern IoT ecosystems. Such static, design-time approaches suffer from scalability issues since they are not suitable for large, distributed systems and their real-time constraints. A new, dynamic approach is required, which requires IoT system design to be reconsidered, enabling properties of autonomy and intelligence. This thesis addresses interoperability from such a dynamic perspective, based on Organic Computing concepts. An architecture for a self-adaptive IoT system will be presented which is able to maintain and improve runtime interoperability in an IoT ecosystem autonomously.

Acknowledgments

I would like to thank the following people who have helped me undertake this research:
My supervisors at the Bosch.IO GmbH, Mr. Patrick Ackerer, Dr. Nikolaos Oikonomidis and Dr. Abdelmajid Khelil, for their support, encouragement and patience;
My colleagues in the Engineering and Digital business department at the Bosch.IO GmbH for their insights and help regarding all IoT related topics;
The members of the Department for Organic Computing chair at the University of Augsburg for their input and support especially on XCS and Reinforcement learning.
And last but not least I want to give special thanks to my parents for always supporting and encouraging me throughout this journey.

Contents

List of Figures

List of Tables

Part I

Introduction and Background

Chapter 1

Introduction

The advent of the Internet of Things (IoT) provides a disruptive potential to the world of communication similar to the disruption of the World Wide Web in the early 1990s. The underlying idea of the IoT is to attach communication technology to physical objects, to become *online* even if they were not initially designed with this capability in mind. This will extend the current reach of the Internet, from interconnected computational machines to cars, buildings and home appliances (among others) and thus embedding technology in every aspect of our modern life essentially blurring the boundary between the physical and virtual worlds. An Internet of Things can be defined as a *"world-wide network of interconnected objects with unique identification and addressability based on standard communication protocols*" [AIM10] which implies a potentially enormous numbers of objects involved. This vision of the IoT promises exciting innovation opportunities in any domain, from smart homes to smart cities to Industry 4.0 and is sometimes extended to the idea of a *"Internet of Everything"* where any combination of sensors, actuators and computing devices are connected with or without humans in the loop [Ini15]. The Internet of Things thus allows people and *things* to be connected "*anytime*, *anyplace*, with *anything* and *anyone*, ideally using any path/network and any service" [VF13].

1.1 Research problem statement

To achieve the vision of the IoT, devices and other desirable IoT resources need to be identifiable and have the ability to communicate and interact with other devices and resources in the global IoT network [MSDC12]. Early development in the IoT space thus focused on the development of RFID tags, sensors and actuators. Also, increased focus was put on the research for wireless and

wired networks, especially with regards to the specific resource constraints of IoT enabled devices which typically operate in low-powered settings.

Early on, the need for IoT platforms has been recognized as a foundation for the IoT [IoT16]. IoT platforms are the logical development arising from the trend of more and more IoT devices and heterogeneous IoT deployments. IoT platforms make it easier for administrators to access and manage IoT resources and for developers to develop services based on legacy technology. Due to their importance in the IoT, there has been a trend to implement IoT platforms of various scale and technological sophistication [IoT16] with nearly every IoT domain and every IoT vendor developing their own platform [Brö17][GPP+16] which resulted in the introduction of over 450 IoT platforms (as of 2019). This naturally led to the development of IoT solutions built in vertical silos, which means, solutions are contained in their desired domains. This means, that device vendors create their own custom solutions by integrating devices into proprietary software and hardware stacks. Since multiple vendors create similar sensors, actuators and platforms, this creates a landscape of *IoT islands* that prevents devices and platforms from communicating among each other due to different standards [GPP+16].

For an IoT solution provider, who does not develop its own IoT platform it is important to choose among these platforms the one which allows for future-proof and efficient development of IoT applications of services. This task is impeded by the abundance of platforms [Brö17]. Not only the amount of platforms but also the missing uniform terminology and comparability makes the choice of the *right* platform difficult. Current IoT solutions are thus still mostly developed as closed systems around vendor-specific platforms with little interoperability between the platforms. This is opposed to the vision of the IoT as a global ecosystem of interconnected devices through the digital infrastructure [AIM10]. The current state of the Internet of Things thus consists of isolated software ecosystem, i.e. IoT applications and services that are restricted to a particular platform ecosystem. Hence it is vital to find solutions that are able to connect these separate IoT silos across platforms, but currently there is no uniform solution available.

In the context of smart homes and smart cities, in recent years the trend went towards more and more IoT providers which enter the market which requires more and more systems to interoperate [Sma18]. Only a small amount of smart home providers offer advanced interoperability support while most providers still refrain to provide information about their APIs which is necessary to connect smart home platforms [Sma18]. The role of IoT platforms in these ecosystems is to provide the services and the infrastructure at the same time.

> *" Middleware platforms provide interoperability capabilities and abstractions over physical devices and services to applications and/or end-users, as well as means of managing the increasingly myriad of IoT devices associated to the systems that use data provided by them "* [MCG+14].

A gap analysis on IoT platforms by [MMST16] identified four aspects related to interoperability. They identified that the interoperability gap in the IoT can be broken down into *"ecosystem formation"* [MMST16] and *"formation of IoT marketplaces"* [MMST16]. *"Ecosystem formation"* deals with the need to consider the current divergent IoT verticals as a coherent ecosystem to provide new and innovative services. Currently it is impossible for developers to integrate multiple foreign devices into their IoT applications. It is still similar to the problem of trying to install a software on Mac that is only available for Windows and vice versa. But, when every piece of data can be read and interacted by every other device and service, the power of the Internet of Things emerges. Vertical isolated systems should be able to collaborate to achieve a higher purpose in different scenarios. Vendors of IoT solutions need a single unified framework for design and development that can interoperate across diverse data environments and under widely differing usage scenarios. However, it is unrealistic that there will be one standard that will prevail in IoT [SZZ+16], which results in the current landscape of multiple disparate standards.
In practice, most existing systems specify and implement interoperability in the syntactic and connectivity categories in detail, dealing with the organizational and semantic categories essentially at the documentation level.

> *"IoT architectures are built on heterogeneous standards, for example, the Constrained Application Protocol (CoAP), Message Queuing Telemetry Transport (MQTT), LightweightM2M (LWM2M), Sensor Web Enhancement (SWE), oneM2M or even proprietary interfaces"* [Brö17]

In contrast with today's applications, where data is usually consumed by single applications, data in the IoT will be shared among different IoT applications, thus requiring a greater interoperability [Bor14].

The second mentioned gap relates to missing IoT marketplaces that *"facilitate the discovery, purchase and distribution of the applications"* [MMST16]. These marketplaces are important as a driver for a vibrant ecosystem of interoperable IoT services, effectively disrupting IoT platform

boundaries. The previously introduced vision of an IoT ecosystem, enabling new business opportunities and value added services, is deeply connected with the availability of open IoT markets. The true value of the IoT lies in an open IoT ecosystem of IoT vendors that allows dynamic, adaptive and environmental-aware applications that are formed of IoT devices and software [Brö17]. In other words, this vision can be described as a composition of independent software services and applications resulting in a so called *systems-of-systems*.

The interoperability barriers regarding difference between hardware and software are particularly visible in the IoT. In order to provide a running IoT solution, an IoT vendor needs to cover the whole IoT stack from the device level to the software level, which creates high costs even for simple solutions. Thus, the market entry barriers are still too high for smaller vendors to sell their products and offer applications and services on top of existing IoT solutions. Also inefficiency in IoT solutions cannot be prevented that results from similar sensors being deployed by different vendors at the same location [SZZ+16]. We can compare the Internet of Things with the early days of the Internet. Imagine it would still consist of unconnected intranets; Facebook, Twitter or Google that connect billions of people today would not have been possible. Interoperability concepts are necessary that can bridge the gap that is increasingly growing between on the one hand cloud based IoT platforms with endless computing resources and systems-on-a-chip solutions that operate under high uncertainty with regards to power, connectivity and computing resources. Since the IoT is developing at a rapid pace, interoperability solutions need to be flexible enough, to also cover future systems. This raises doubts concerning standards based approaches, since the process is generally too time consuming for the fast pace of IoT development and is bound to the design time of systems.

In current IoT ecosystems we observe the following interoperability related problems which occur repeatedly:

- Interoperability problem 1: Identifying the right interoperation partner is only possible when the partner is known at design time and the interface is known. If the partner fails or disconnects at runtime, the partnering application will fail to operate since alternative partners can not be integrated without manual effort.

- Interoperability problem 2: Data formats between IoT applications and services are usually exchanged in their native formats through some standardized communication protocol

and format (e.g. XML/JSON). In the IoT, due to its heterogeneous device data and domain penetration, this results in incompatible data items. Different IoT services are not able to automatically process this information, thus approaches that introduce an abstraction level are necessary. Ontologies and semantics look to be promising in this respect [Bor14].

- Interoperability problem 3: A manual integration approach of multiple IoT applications leads to tightly coupled applications which results in failure prone and rigid application structures. If the interface specification of the interoperation partner changes, manually changing the integration code is necessary which results in high costs, inflexible agreements of semantics, standards & QoS. Therefore it is cumbersome, for example, to exchange services at runtime. Besides, if the interface description on one of the systems in the collaboration changes, the consuming system is not functional anymore (except the providing systems does not offer any functional related data). This prevents the development of ad-hoc, innovative solutions and also cause problems when sub-systems versions change (what they will eventually do). Also, extensive testing is necessary since automated testing is difficult in manual integration approaches [CFMP05].

If we can solve these issues, we can envision the future Internet of Things consisting of a complex system of systems where systems of different scales and capabilities are connected in a cross-domain, cross-platform manner to create new applications that perform tasks in order to improve human life. Interoperability acts as the *"glue"* between these disparate systems. It enables new gains in terms of business and technology and serves as an enabler for digital IoT ecosystems.

This thesis will focus on the interoperability aspects of IoT platforms and to which extend existing interoperability solutions from the IoT domain and other domains already address different parts of these aspects and what is still missing. In particular, the following research questions are in focus:

1. What is the current state of the art of interoperability in IoT ecosystems?
2. What are the reasons that existing interoperability solutions did not yet solve the problem of interoperability inside the IoT space?
3. What is a proper concept of interoperability in IoT ecosystems?
4. What are the requirements to establish interoperability in IoT ecosystems?
5. How do IoT systems need to be designed in order to implement this model of interoperability and thus addressing the requirements for interoperability?

By answering these questions the thesis will contribute to existing interoperability research by providing a theoretical model of runtime interoperability as well a practical architectural reference to implement interoperability solutions in existing IoT systems. In particular it will show that interoperability between IoT platforms is essentially a Systems-of-Systems problem and can thus be described using terminology from Systems-of-systems and Digital service ecosystems research. Since this is a unique perspective on the interoperability problem it will provide a thorough analysis of the interoperability deficits of existing solutions to derive a formal theory of runtime interoperability. This formal theory helps system analysts and designers to understand and plan according to the unique interoperability problems in IoT solutions since it covers all the major aspects to be considered. Especially, semantic and pragmatic interoperability will be presented as important concepts of which pragmatic interoperability has previously been neglected in interoperability research. For the IoT practitioners a self-adaptive IoT reference architecture is proposed and evaluated to proof the benefits of designing runtime interoperability solutions in practice. To aid in this process, proper metrics are introduced as well, to make the interoperability problem measurable. The reference architecture will also demonstrate the merging of two, before seemingly unrelated, areas of software and systems engineering: IoT and Organic computing. The thesis will present the benefits of applying Organic Computing research knowledge to overcome the IoT interoperability gap and showcase the advantages compared to previous IoT interoperability solutions.

1.2 Conceptual approach and thesis structure

To address the research questions, the following methodological approach is pursued (figure 1-1)

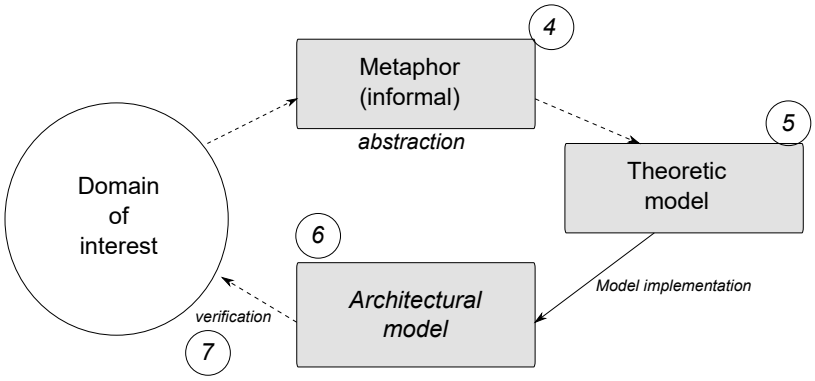


Figure 1-1: Methodological approach for this thesis. The numbers correspond with the chapters.

In chapter 2 the necessary background knowledge for a conceptual model on IoT interoperability will be established. Chapter 3 will summarize existing work in the context of IoT platform interoperability before in chapter 4 an informal conceptualization of IoT ecosystems and interactions in IoT ecosystems is presented. This conceptualization forms the basis for the next contribution - a theory of runtime interoperability in IoT ecosystems in chapter 5. The runtime interoperability theory models the runtime interoperability problem which gives input to the third contribution in chapter 6: a model architecture to implement the runtime interoperability theory. The implementation of this model will be evaluated in a feasibility study in chapter 7. The final part of the thesis comprises a critical discussion of the results and an outlook for future work.

Chapter 2

Background

2.1 Aim

Research on Interoperability and the Internet of Things has spawned a large body of theories and concepts through various disciplines in the recent past. This chapter will give a general overview about the state of the art in the Internet of Things and interoperability research sector. This is important in order to understand, first of all, the problem of interoperability between IoT platforms more clearly and also to grasp the lacks of existing solutions for cross-platform interoperability. The first section present a summary about the most important concepts in the IoT space followed by an introduction to the concept of "cross-platform interoperability" and why it is important. Then, the necessary background around the complex topic of interoperability is addressed.

2.2 A definition of the "Internet of Things"

> *" The Internet of Things can be defined as a global information infrastructure enabling advanced services by interconnecting physical and virtual things based on interoperable information and communication technologies"* [Gmb16]

The first origins of the IoT can be found in M2M telecommunications and RFID technology, with the latter even dating back to the second world war [Gmb16][Ini15]. RFID technology largely considers the remote identification of objects through radio frequency communication systems, with M2M being a generic term to comprise all kinds of telecommunication technologies to identify and control remote machines. In the past, these M2M systems were built by large organizations, such as oil & gas companies or other well-financed organizations due to the high costs and expertise

required [Gmb16]. However, in the recent decades, M2M technology has spread into more and more industrial and consumer sectors. So why do we experience this transformation of originally seperated M2M solutions into world-wide networks of devices, interconnected through the Internet? According to a recent report by [IoT16], there are several reasons for this. First of all, the cost of hardware has decreased over the last decade while at the same time hardware has become smaller and more powerful. This has increased the trend towards ubiquitous computing, where we find sensors and actuators in more and more everyday devices and products. This has positioned the IoT from originally industry/vertical specific solutions into the mass consumer markets since more and more vendors advertise their products as being *"IoT enabled"*. The widespread use of Internet services has further accelerated the mass market adoption. In addition, Big data and cloud platforms provide the necessary infrastructure to handle the increasing load of IoT sensor data. The core elements of M2M, remote monitoring and control, data collection & analysis have found applications in logistics/ industrial automation/ health care / city administrations etc. [Gmb16]. To draw a line between M2M and the IoT is challenging, therefore these terms are sometimes used synonymously. Generally M2M solutions restrict to vertical domains while IoT solutions focus on integrating sensors and information systems from multiple domains to cross verticals [Gmb16]. The Internet of Things could be described as an evolution of the M2M sector with the focus on crossing vertical boundaries and gaining insights from multiple sources for the purpose improving or customizing products and services or supporting decision making [Gmb16].

> *" The Internet of Things (IoT) in its essence describes how the physical world is being connected to the Internet "* [IoT16].

The *Things* in IoT refers to any physical object that is relevant from a user or application perspective and is uniquely identifiable. The Internet serves as the network to connect these physical objects together.

> *" The interconnected things have physical or virtual representation in the digital world, sensing/actuation capability, a programmability feature and are uniquely identifiable. The representation contains information including the thing's identity, status, location or any other business, social or privately relevant information. The things offer services, with or without human intervention, through the exploitation of unique identification, data capture and communication, and actuation capability. The service is exploited through the use of intelligent interfaces and is made available anywhere, anytime, and for any-*

thing taking security into consideration" [Ini15].

Ubiquity is a major feature of an IoT system, indicating a network which is available anywhere and anytime. But in the context of IoT, the concepts of *"anywhere"* and *"anytime"* need not necessarily refer to *"globally"* and *"always"* but rather to *"where"* and *"when"* it is needed [Ini15].

2.2.1 Related concepts

The concept of an "Internet of Things" was not *"invented"* but rather emerged from different, related disciplines in the Computer science / systems engineering and Information systems. Most importantly, these are: cyber-physical systems, ubiquitous and pervasive computing.

Cyber-physical systems Cyber-physical systems and the Internet of Things are two concepts closely related, sometimes used interchangeably. As with the IoT the idea behind cyber-physical systems is to embed the physical environment with computing and communication capabilities in order to change the way humans interact with the world [RLSS10]. Examples for such systems can be found in all domains, from healthcare to industry , from buildings to military systems. The every increasing trends of smaller and smaller form factor microcontrollers and sensors, improved energy efficiency and abundant network bandwith and connectivity are the driving forces behind cyber-physical systems. They are build out of an amalgamation of embedded systems, real-time systems, distributed sensors and controls connected through a network backbone [RLSS10]. Cyber-physical systems are formed to build clusters of wireless or wired networks based out of sensors and actuators [RLSS10], the same principle underlying the Internet of Things concept. The promise behind this lies in new and innovative solutions that will solve essential present and future sociotechnical problems. Due to the closeness of the two terms, this thesis will not differentiate between CPS and IoT systems and assume that concepts for the Internet of Things are similarly applicable to cyber-physical systems.

Ubiquitous computing and pervasive computing Two other concepts that are related to the Internet of Things are ubiquitous and pervasive computing. Compared to cyber-physical systems, they describe a broader view on the merger between computing technology and society. "Ubiquitous computing" was coined in the beginning of the 2000s, as a term for the increasing trend that computing devices become more and more embedded in everyday life and at the same time seem

to "disappear" since the are simply available anytime, anywhere without effort [LY02]. The trend started with computing machines becoming smaller and smaller in form factor while at the same time becoming more and more capable for complex computations, resulting in a society where *"computers will be embedded in our natural movements and interactions with our environments - both physical and social"*[LY02]. Ubiquitous computing can be understood as combining the areas of pervasive and mobile computing, where mobile computing is concerned with the increased mobility of computing devices and pervasive computing which is the capability of computing devices to sense the environment and use the sensed information to build computing models of its surroundings to become "intelligent" [LY02]. This defines ubiquitous computing as *"[...] any computing device, while moving with us, can build incrementally dynamic models of its various environments and configure its services accordingly"* [LY02]. As it was shown by the smartphone trend since the late 2000s, computers with large degrees of computing power comfortably fit into a pocket and are able to assess and interact with the environment, essentially manifesting the idea behind ubiquitous computing as a prime example.

Cooperative objects Various research projects in the past have already achieved tremendous breakthroughs in the area of wireless sensor networks, IoT and CPS solutions. One of them is the Cooperative Hybrid Objects in Sensor Networks project (CHosen)[HS11] driven by the European commission which was operated between 2008 and 2011 with the goal of designing wireless communication technology for highly challenging domains in aircrafts and automotive [HS11]. The main outcomes of the project were a new RF transceiver concept, a new class of wake-up receiver and a new protocol processing unit [HS11]. The project could demonstrate that these new technologies were able to overcome the demanding challenges in the named domains. This is a prime example for the need for further research in the IoT space, as the pervasive nature of connected solutions also has to cope with the various, specific domain requirements. However, summarizing all of the extensive research body in the IoT sector is beyond the scope of this thesis. Relevant projects to the topic of interoperability in the IoT will be described in the proceeding chapters.

2.2.2 Sociotechnical aspects

Figure 2-1 expresses areas that are influenced or influencing the IoT as defined by [Ini15]. Enabling technologies for the IoT contain hardware, gateways, protocols and sensor networks. These enabling technologies form the foundation on which services and applications are built for specific

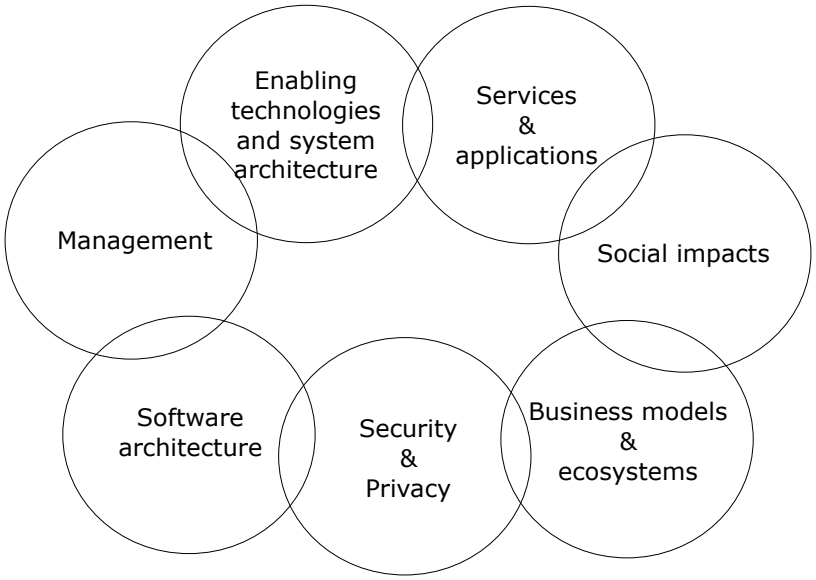


Figure 2-1: Social and technological aspects affected by the IoT [Ini15]

domains (smart cities, smart homes,...) using software architectures and technologies, in particular APIs, cloud computing and middleware. The IoT enables entirely new business models and the creation of new ecosystems and value chains. It's social aspects lead to changes in the ways people interact and live with technology. However, this puts new challenges on security and privacy, since technology is even more integrated in our everyday lives.

2.2.3 IoT characteristics

[RMjP15] distinguish between IoT characteristics for IoT infrastructure and IoT applications. Generally, the IoT infrastructure, as already described, can be made up of the heterogeneous and resource constrained devices from multiple vendors that are integrated with traditional computing hardware through the Internet. The IoT makes up an ultra large scale, dynamic network of connected *Things* with unique characteristics, from very small RFID/NFC tags to high-end computing devices. An important aspect to consider (compared to other Internet service) is that since *Things* are real world objects, they exhibit spontaneous interactions with other *Things* since a large number of devices are mobile and move around thus coming in contact with other devices [RMjP15]. Figure 2-2 illustrates the large heterogeneity of devices that appear in the IoT space as outlined in [RMjP15]. *Things* are embedded with sensors and actuators to perform the sensing/actuation which enables the *smartness* of the IoT. The *Things* have communication capabilities embedded, usually based on standard and low-level communication protocols in order to be networked among

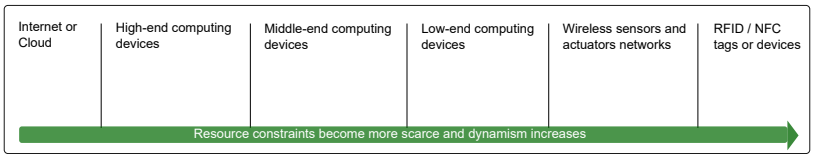


Figure 2-2: Device heterogeneity in the Internet of Things, from cloud level to device level [RMjP15]

another. They are furthermore programmable.

> *"At the simplest level, a programmable device is one that can take on a variety of behaviors at a user's command without requiring physical changes whereas on a higher level, devices can be programmed to control other devices or systems autonomously. The scope of an IoT system varies from a small system which contains uniquely identifiable things and small sensors to a system that interconnects millions of things with a capacity to deliver complex services"* [Ini15].

IoT applications are built as a composite of traditional software and *Things*, affecting the state of the real world. They can be characterized, according to [RMjP15], as diverse and real-time sensitive. IoT devices are used in various domains as part of IoT applications, hence the space of application is very diverse. Applications can be built to provide real-time services since IoT devices offer real-time sensing, or aggregate IoT data in a non real-time setting. As more and more *Things* are connected, more services will be built on these devices, thus transforming monolithic applications into service agglomerations that are reusable in different contexts. The exposure of real world devices to the Internet increases the security and privacy attack surface even more compared to the already large surface with smartphones and PCs. Hence security and privacy are important areas of ongoing research and technological advances in the IoT space.

2.2.4 The architecture of the Internet of Things

The following figure 2-3 provides an overview of the state of the art IoT reference architecture according to the reference architecture presented in [GBF+18].

Devices (i.e. *Things*) serve as the connection between the physical and the virtual world, since they are on the one side connected to sensors and actuators via drivers and on the other side provide processing and connection capabilities to the IoT middleware [GBF+18].

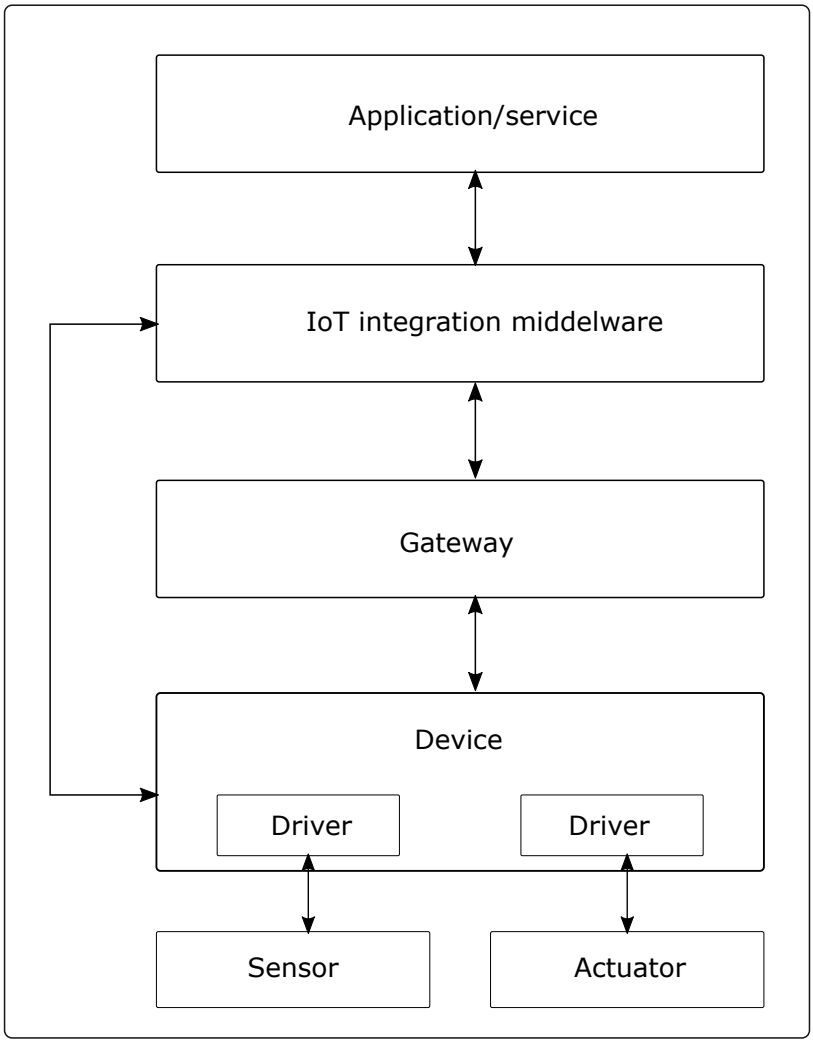


Figure 2-3: The IoT reference architecture from application/service level to device/sensor level [GBF+18]

A **gateway** provides a way for devices to connect to other systems if they do not possess the capabilities themselves. IoT gateways have various IoT protocols implemented and thus can format communication between devices and other systems [GBF+18].

The **IoT integration middleware layer** (or simply IoT middleware) is a *"software artifact between the application layer and the infrastructure support (communication, processing, and sensing) offering a standardized means to access data and services provided by the smart objects via a high level interface"* [Bat13]. Specifically it offers (i) *"receiving data from the connected Devices"* [GBF+18], (ii) *"process the received data"* [GBF+18] , (iii) *"provide the received data to connected Applications"* [GBF+18] , and (iv) *"control devices"* [GBF+18]. Such a middleware also promotes the reuse of generic services that can be composed and configured to make easier the development of applications in a highly distributed and heterogeneous environment. Middleware solutions are common methods to improve interoperability (on various levels), since specifics of the technological layer, such as heterogeneous protocols, formats or other details can be abstracted towards the application layer. This makes it easier and faster to create new applications on this stack. On the highest level, applications and services are built based on the middleware services, gateway and devices. The IoT reference architecture is usually implemented through **IoT platforms**, hence the next section will give a thorough introduction to the IoT platform concept.

2.3 IoT Platforms

Increasingly complex IoT solutions require more advanced communication platforms and middleware that provides seamless integration of devices, networks and applications. This has led to a considerable increase in development activities for IoT platforms in the recent years. IoT platforms are a new kind of software platform, specifically tailored for the characteristics of the IoT [IoT16]. IoT platforms are a relatively new development and feature a great diversity in terms of functionality and sophistication [Gmb16]. Thus, over 450 [IoT16] platforms have been developed over the years which cover different parts of the IoT reference architecture. The advantages of using an IoT platform are reduced cost and faster development times due to standardized components for *typical* IoT problems, such as device connectivity, remote management or digital identity management. An IoT platform represents a software artifact of a digital platform that is usually diverse and specialized in its application scope. In very general terms, an IoT platform is a software

backend that can be seen as the central backbone of an IoT infrastructure. Usually IoT platforms host a list of common services that are used either by IoT applications, such as database solutions, device connectivity services or externally facing API gateways. The services are typically delivered over a Wide Area Network (WAN) or the Internet using IP based transports such a TCP or UDP or higher level protocols using these as the underlying transport. Thus the services that constitute a platform are essentially subsystems of the platform that work together to offer the platforms capabilities. They facilitate the connection between the embedded hardware in the IoT (sensors, actuators, microprocessors) and the application level by providing interoperability and connectivity among billions of connected devices. Hence they form an integral part of the IoT. This explanation explains why "IoT platform" and "IoT middleware" are usually used synonymously.
IoT platforms offer more than just connectivity. Furthermore they provide services for device management, action management, analytics & visualization and integration with external interfaces and web services. The platform allows to integrate data from multiple smart objects and aggregate and clean that data before providing it to the outside via interfaces. An IoT platform offers software components in order to enable interaction with Smart Objects, to access or manipulate information or to control them. This functionality is oftentimes referred to as services. Moreover, the platform monitors/manages and controls various types of endpoints. Also, it can feed its services with the data from the smart objects, but also with data from other platforms. This flexibility and power makes IoT platforms the key asset to unlock the real value of IoT, since smart objects themselves are rather a utility due to their limited performance and capabilities. Only through the combination of multiple smart object sources, useful services can be created. [IoT16] describe the following, general capabilities of IoT platforms:

- Provisioning and management of IoT endpoints (things) and gateways
- Customizing and building applications (software development kit [SDK], application server, integrated development environment [IDE] and others)
- Event processing: event stream and data aggregation, stream analytics, storage and management, and information management (often collectively referred to as "data digestion")
- Decision processing: rule engines, orchestration of workflows and business process management (BPM)
- Analysis: IoT data analysis and visualization (including dashboards)

- Cybersecurity: authentication, encryption, certificate management (among others)
- IoT device communications (such as MQTT or HTTP)
- Integration: API publish and subscribe, protocol hopping, scalable transformation and adapters to connect to business applications and data sources, cloud services, mobile apps, legacy
- Providing protocol adapters
- Providing user interfaces for both end users and developers

IoT platforms are a software-oriented manifestation of the well-known *platform* concept [GC13], and are a manifestation of the *digital platform* concept [CHP18]. Hence, it is worth to provide a brief introduction on the topic of (digital) platforms first to understand the context and origins of IoT platforms.

2.3.1 Platforms

The interest in the development of platforms in the digital age has risen significantly in the recent years [CHP18] which triggered increasing research interest. The term *platform* is used in so many different areas with fluid interactions between research domains such as Computer science, information science, and economic research. This thesis deals with the area of digital platforms, more specifically to digital software-based platforms which incorporate various modules that extend the functionality of a software product [dRSB17]. Digital platforms are omnipresent in the current digital age [dRSB17], from the popular Arduino hardware platform for building small hardware devices, to Apple's IOS platform which runs applications on Apple IPhones, to the ITunes marketplace which serves a platform to distribute applications to customers inside the Apple ecosystem to Amazon's AWS platform which serves the most advanced on-demand computing platform to date. Digital platforms exist on different levels and in different scales, from these technical software platforms to peer-to-peer digital platforms such as Uber and AirBnB. They have spawned new business opportunities mainly evolving around platform business models. At the same time, research on platform-related topics has matured and experienced increasing formalizations [dRSB17].
Still, because the platform concept is used in a lot of different domain and contexts, it is impossible to find an all comprising definition of the platform concept since it would need to be so general, that it would be utterly useless. Thus it is always important to consider, in what type of context the

term *platform* is used. In very general terms, a multi-sided platform mediates different groups of users, such as buyers and sellers. Platforms exhibit network effects, which means that the usefulness of the platform increases with the number of users of the platform [dRSB17]. Digital platforms can be considered as multi-sided business platforms, where we will focus on the technical aspects of digital platforms, considering them as technical artifacts. In technical discussions, one often speaks of *computing platforms* or more specifically of *software platforms*. Computing platforms however are also referred to on different abstractions. The most intuitive example of a computing platform is an operating system (OS) that abstracts underlying hardware by offering generic commands to the developer of software on this particular OS. Common examples for computing platforms have become Cloud Computing / PaaS offerings, software frameworks or virtual machines such as Java or .NET. A definition of a digital software-based platform which comes closest to the idea of IoT platforms is provided by [TKB10]:

> *"Software-based external platforms consisting of the extensible codebase of a software-based system that provides core functionality shared by the modules that interoperate with it and the interfaces through which they interoperate"* [TKB10]

IoT platform can be considered a concrete expression of a *digital software-based platform* [dRSB17].

2.3.2 Types of IoT platforms

Similar as with the distinction inside digital platforms, one can distinguish different levels of IoT platforms. The IoT platform stack usually covers three general levels as described in [WF15]: The *"device level"*, *"connectivity level"* and *"application level"*. IoT platforms are accordingly clustered into: (i) *"device level platforms"*, (ii) *"connectivity management platforms"* and (iii) *"Application enablement platforms"* by [Gmb16]. But, this does not mean that platforms **only** implement one type. Rather, more sophisticated platforms usually cover all levels, from device to the application/cloud level.

Device management platforms Starting from the bottom of the IoT platform stack, the "device level" connects *"IoT-specific hardware, such as sensors or actuators and embedded software which is used to integrate new devices or operate the functionality of the physical thing"* [WF15]. Device management platforms sit on top of hardware/devices and allow to manage them remotely such as upgrading software, checking health and/or adding or removing a device.

Connectivity management platform Connectivity management platforms CMP provide a way for devices to connect with each other (local network of all sensors or devices in my home or office or industry) and/or when they directly or through a controller or gateway connects to Internet using standard communication protocols, such as MQTT or LORA [WF15]. They also offer device and subscription management features. According to [Gmb16], the market of CMPs has featured a platform diversifications with dedicated solutions for different markets.

Cloud level / application enablement IoT platforms The highest sophistication of IoT platforms is reached at the "cloud level", also referred to as *"IoT application enablement platform"* [IoT16], which solves the main tasks of bringing together different communication streams and devices for rapid IoT application development and provision. The name "application enablement" already hints to the core value proposition of cloud level IoT platforms, i.e providing common horizontal components that can be re- used across industries and market segments and tools, frameworks and APIs for event processing and business logic implementation [Gmb16]. Hence, cloud level IoT platforms are the most prominent type of IoT platform due to their ease of development and non-existing resource constraints [MMST16]. They are provisioned in terms of PaaS offerings (provide cloud computing services for IoT devices and data, e.g. storage facilities, device management, device connectivity,...) or SaaS offerings, focuses on the mashup of data using cloud computing capabilities. [MMST16] further identify the following trends among cloud level IoT platforms:

- Platforms commonly use REST APIs. Current IoT services will tend to become more and more like traditional web services
- APIs support interaction with the connected devices on the platform as well as the management of these devices
- IoT service mashups and data analytics will be key integrators for the future of IoT technologies
- Few platforms have service discovery mechanisms
- Data ownership is a common problem

Since "cloud level" platforms are the most complex breed of IoT platforms and IoT platforms are not standardized, a large variety of architectures have been developed, from the most simple ones

just providing connectivity, to full-fledged platforms such as the *Bosch IoT suite* [1] or *ThingWorks PTC* [2]. Hardly any vendor can develop a true end-to-end IoT platform able to support **any** major vertical industry or application today. Most vendors are focusing on specific market segments, or rely on partner ecosystems to provide the full set of components necessary to enable a given customer solution.

The advantages of using IoT application enablement platforms for development in the IoT are manifold. First of all, applications that need to connect to hundreds or thousands of different devices from different manufactures can make use of IoT platform's device management features. This spares the developer of an IoT application of creating that infrastructure himself. Since the main purpose of an IoT application is to connect with IoT Things, this building block is essentially a part of every IoT solution, thus all IoT platform providers offer this functionality. Also integration of database technology and application logic UI is facilitated by using platform standard solutions. IoT analytics is also an important and specific part to IoT solutions and platforms offer common libraries and frameworks with pre-built algorithms that reduce the amount of time, comparing to building such a system from scratch significantly. The platform furthermore provides the capability to monitor IoT event streams; enables data aggregation, specialized analysis and application development; and engages back-end IT systems or services. It typically plays a vital role in providing functionality for provisioning, **controlling and even changing** the endpoints to support IoT solutions. The distributed IoT platform responsibilities may be fulfilled, in part, near the devices or in a public or private cloud.

Current IoT platform landscape The current landscape of IoT platforms is highly sophisticated and diverse. There is a wide range of IoT platforms available, distinguished into applications, enablement and building blocks. A comprehensive overview of available IoT platforms is outside the scope of this thesis, hence only some of the most popular IoT platforms are presented based on the analysis already performed and cited by [GBF+18]. For a more detailed analysis of current IoT platforms, the reader is referred to [GBF+18].

FIWARE [3] is a middleware platform and open source framework whose development has been sponsored by the EU and Europen commission [GBF+18]. It consists of 52 components in 6 groups. The platform is built based on an OpenStack based cloud hosting rich library of modules

[1] BoschIoTSuite
[2] ThingWorksReference
[3] https://www.fiware.org

offering various added-value services. These modules (termed Generic Enablers (GEs)) fulfill all the required capabilities of an IoT platform, i.e. device management, device discovery and brokerage. A gateway is responsible for managing the communication of devices with the IoT backend. The data context broker is another important component of the main functionality of FIWARE, to connect further components to the platform .

OpenIoT [4] is an open source, middleware infrastructure providing the the collection and processing of data from virtually any sensor in the world, including physical devices, sensor processing algorithms, social media processing algorithms and more. OpenIoT facilitates the following essential IoT-related activities: the integration of sensors; semantically annotating sensor data; streaming data of various sensors to a cloud computing backend; dynamically discovering/querying sensors and their data; composing and delivering IoT services that comprise data from multiple sensors; visualizing IoT data based on appropriate mashups (charts,graphs, maps etc.); optimizing resources within the OpenIoT middleware and cloud computing infrastructure. These core activities of OpenIoT correspond with the basic IoT platform stack as shown earlier. The OpenIoT platform is universally applicable in different domains, with a particular focus on providing efficient ways to use and manage cloud environments for IoT entities and resources such as sensors, actuators and smart devices and has been validated in different environments (CSIRO, Fraunhofer IOSB).

The Bosch IoT Suite [5] is an open standards and Open Source based platform as a service (PaaS) to realize cross-domain applications (among others in the domains of Smart Home, Smart City, Connected mobility and Industry 4.0). It has been released in 2014 and is still actively developed, with more than 250 customers operating on the platform. The core components of the Bosch IoT suite are : Bosch IoT Analytics for analytic services on top of device data, Bosch IoT Hub for the integration of devices into the platform, Bosch IoT Permissions for permission handling , Bosch IoT Remote Manager for performing remote updates of IoT devices , Bosch IoT Rollouts for performing over the air firmware updates and Bosch IoT Things for device management We can observe, that these core components refer to the IoT platform stack.

[4] http://www.openiot.eu
[5] www.bosch-si.de, accessed on 24.12.2019

2.4 IoT platform ecosystems

A platform ecosystem can be defined as: *"the extensible codebase of a software-based system that provides core functionality shared by the modules that interoperate with it and the interfaces through which they interoperate"* [GC02] Platform ecosystem formation has been identified as one of the most pressing gaps in the IoT at the moment by [MMST16] in order to create a socioeconomic environment of buyers, suppliers and makers that create IoT applications and services. While the early stage of the IoT has largely taken place in closed, isolated environments, the recent trends and developments [MMST16] have given rise to the assumption that this trend is changing and there is a continuing interest towards large scale/open environment scenarios. In [MMST16] the authors expect *"ecosystem formation"* through the following achievements: (i) *"easily expandable platforms"*, (ii) *"cross-platform sharing of applications and services"* and (iii) *"local composition of services"*. A motivation for such an ecosystem formation activities can be seen in the IoT scenario of - *smart parking*. In the smart parking scenario the goal is to connect physical parking spots, why parking sensors with apps or vehicle control units in cars so that the parking journey for customers is enhanced. A car manufacturer does not operate an infrastructure of parking spaces in cities but builds applications for embedded devices inside cars that need to contact parking spot - providers in different cities. The same applies for an App developer who distributes a "smart parking" app. Since each city usually has their own IoT platform for managing parking spots in place, the manufacturer and/or developer needs to integrate with many different parking platform operators so that the autonomous cars (or the app) can use information about parking spots from multiple platforms. But forming bilateral contracts between each and every parking spot provider does not scale, which is why a such open IoT platform ecosystems are necessary.

Closed and open IoT ecosystems Current IoT platform ecosystem largely do not fulfill all (or any) of the requirements proposed by [MMST16]. This is generally because of silos of platform-centric solutions and the inability to aggregate data of multiple platforms into a single application [MMST16]. Also, proprietary platforms, as opposed to open source platforms, do not allow to add reusable components or add-ons to the platform [MMST16]. Consider for example the case of a smart production system which is a use case from the Industrial IoT domain, visualized in figure 2-4.

As shown in the figure, the industrial IoT environment is occupied by a multitude of stake-holders such as industry service providers, plant operators, machine providers and customers but

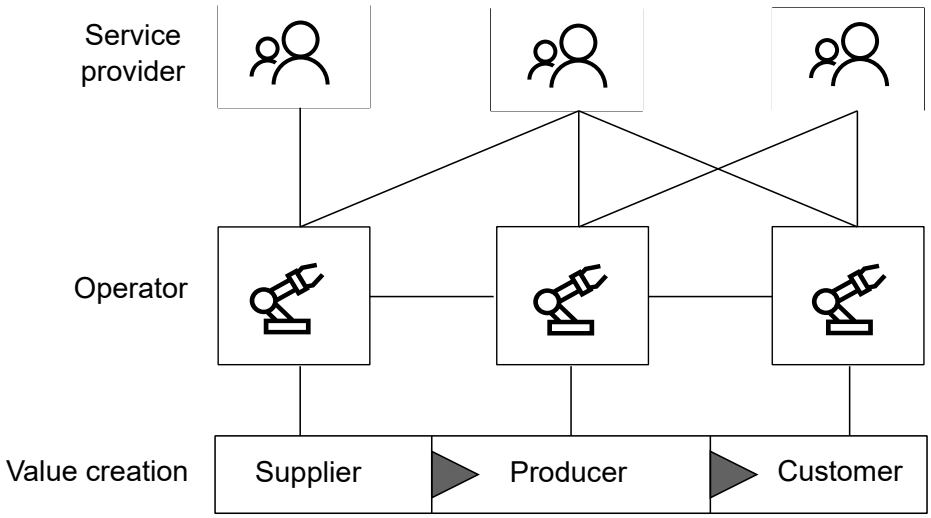


Figure 2-4: Example for an industrial IoT ecosystem of multiple machine contributors, contractors and customers, based on [aH15]

also new roles such as intermediaries. Due to this heterogeneity, multiple ecosystems develop, inside and across company borders [aH15]. But, as per the current Industry 4.0 / Industrial IoT state, the desired properties for IoT ecosystem formation, as postulated by [MMST16] do not hold. There are information silos alongside the horizontal and physical value chains [aH15], since plant operators manage business partner manually and bilateral and do not exchange data. Operational machine data is thus not used sufficiently enough. There is a low transparency about available data offerings and service offerings in the market, making cross-platform sharing and composition of services difficult or even impossible and leading to silos of platform-centric solutions, as described by [MMST16]. More generally, we can identify that the IoT is diverse and includes *"individual entrepreneurs, small communities, public sectors and large organizations from large industries (some of them with leading roles in the second and third industrial revolutions)"* [NHRdR18] with diverse organizational logics. Open IoT platform ecosystem would allow for the exchange of platform data and services across platform boundaries through service platforms where intermediaries act as trusted instances for collaboration inside the ecosystem [aH15].

IoT marketplaces Intermediaries are often assumed to take on the role of **marketplaces**, to discover, share and purchase these IoT data and services. A marketplace is defined by its allocation and pricing rules, and its technical infrastructure and business models (e.g., providers have to pay a fee to participate in the market). Marketplaces allow for monetization opportunities which

are currently still missing from the IoT [MMST16], and hence inhibit ecosystem formation and building sustainable IoT solutions. IoT marketplaces address various customer demands, on the one hand they offer application and service providers a platform to share their services and reach a large target audience on the other hand they offer IoT customers to find providers for services among a large range of IoT platform providers.

> *"From the perspective of a service provider who offers one or multiple services on such a marketplace, the question is which service configuration and price it has to offer in order to be allocated to consumer requests. This is a complex problem, as it not only depends on the specifics of the requests, but also on the offers and configurations of competitors."* [CN15]

Marketplaces form small economies which are mainly governed by supply and demand and thus exhibit their own market dynamics. The main difficulty with marketplaces usually stems from the *"chicken-egg-dilemma"* which in simple terms deals with the problem, that a marketplace needs a minimum amount of momentum of consumer-provider interactions to get started. If there is no supply on the marketplace, no consumer will join and if there are no consumers to buy a product, no provider will offer their product on the marketplaces.

Although the dynamics of marketplaces are a fascinating topic in their own right, this thesis focus does not focus so much on the business aspects of marketplaces but more on the technological aspects, thus referencing the so called *"Software application marketplaces"* [MMST16]. However, overlaps to the business domain are not inhibited. Indeed, one can also conclude, that the technical and business oriented definitions have some overlap, as the technical platform also connects consumers and producers, i.e. developers and resources.

Ecosystem formation Open ecosystems and cross-vertical, cross-value chain collaboration are crucial in the IoT because much of the proposed innovation and value is based on integration of data from diverse sources [Luc16]. Concretely, in the Smart production use case, this leads to production optimizations for the plant operator and increased customer satisfaction, since cross-company data can be analyzed and used for example for services such as *predictive maintenance* [aH15]. However, missing interoperability between IoT platform providers prevents the benefits of service co-creation to be reached. Hardly any vendor can develop a true end-to-end IoT platform able to support such cross-vertical services and applications today. Most vendors are focusing

on specific market segments, or rely on partner ecosystems to provide the full set of components necessary to enable a given customer solution. There are a range of generic functionalities that application enablement platforms should support to enable rapid development of IoT solutions that can adapt to the customers' evolving business requirements.
As explained before, the dynamics inside the IoT platform ecosystems generally tend towards cooperation and joint service co-creation. Because of the close relation between IoT platform ecosystems and general platform ecosystems, literature has discussed challenges in the formation of these ecosystems for a long time. One of the main challenges for the creation of these IoT platform ecosystems is **interoperability**. Interoperation in the IoT will involve cooperative interactions among various heterogeneous systems that satisfy purposes, goals, or mission objectives that are shared by the participating constituents.

2.5 Interoperability

In this section the problem of interoperability, in particular in the context of IoT platforms, which was introduced in the previous chapter is explicated in detail. Interoperability and methods to solve it are the main focus area of this thesis. The section thus gives a thorough introduction to interoperability as it has been researched for decades in various disciplines and plays an increasingly important role in IT and the Internet of Things in particular. Relationships between systems become increasingly necessary, as more and more systems are deployed and required to operate in the same space or even co-operate, for example in Smart Cities or Smart Home environments. Interoperability is a special form of relation between systems that allows them to work together [CFMP05]. For the realization of a true vision of the IoT, one major challenge is achieving interoperability between the various IoT enabling technologies. Additionally, the main issue is not only in simply building an IoT system that connects various IoT devices together, but in maintaining scalable, private, secure and trustworthy operations on the IoT.

2.5.1 Defining interoperability

Defining interoperability is a difficult problem, comparable to defining the Internet of Things. Since interoperability has a long history in systems and software engineering, there exist a broad range of available definitions in literature, that range from very generic to very domain specific ones.

> *"In a very general sense, Interoperability can be defined as to allow some form of interaction between two or more systems so as to achieve some goal without having to know the uniqueness of the interacting systems"* [Asu10].

According to the official IEEE definition[1], interoperability is defined as: *"the ability of two or more systems or components to exchange information and to use the information that has been exchanged"*. A more detailed definition was given by [CFMP05]:

> *"The ability of a collection of communicating systems to (a) share specified information and (b) operate on that information according to a shared operational semantics in order to achieve a specified purpose in a given context. To interoperate, a system must provide a service that is used by another"* [CFMP05].

Traditionally, interoperability is considered from a communication centric perspective. This however is a rather limited approach. In its purest form, interoperability is of course highly connected to communication, there are more factors to consider. As suggested in the following, more general, definition: *"An interoperability problem appears when two or more incompatible systems are put in relation. Interoperability per se is the paradigm where an interoperability problem occurs"* [NLGC10]. One can distinguish between a systems structure and systems behaviour when considering an interoperability problem. The structure defines the system's organization and relationships. The behaviour describes how the systems acts and reacts, where most often the structure defines the behaviour [NLGC10]. Since interoperability is a problem that pervades multiple domains such as enterprise systems, military systems, medical systems, systems theory has spawned a large number of research on interoperability. In [NLGC10] a generic, scientific approach to formalize interoperability based on systems theory is created in which the authors also refer to different facets of interoperability. Their contribution is an ontology that describes interoperability problems in a formal way and provides a framework for describing problems and related solutions pertaining to the interoperability domain. It builds on two models: a systemic model, for which they provide definitions of each important concept; and a decisional model that provides the basis to draw conclusions regarding problems occurring on systems [NLGC10]. The ontology is not domain specific and thus considers the interoperability problem from a systems point of view. They assume, that interoperability is a requirement inside a system and its maturity depends on the interaction or composition of its elements. This means, in order to reach the different levels of interoperability,

[1] IEEE.org

certain properties need to be enabled in order for systems to work together. A very detailed definition from the systems theoretic standpoint is found in [CD12], which is also form the basis for the interoperability considerations in this thesis:

> *"Interoperability of a system S is the set of abilities and capabilities of S to interact with other interfaced systems requesting (resp. providing) services or products provided (resp. requested) by S with the help of enabling systems. This interaction is requested in order to fill, in harmony, a common mission reflecting the expectations of all the stakeholders in any situation and possibly for a limited time. Achieving these capabilities and abilities should not induce inappropriate effects or feared (failures, risks, failure to satisfy other functional or non functional requirements)"* [CD12]

More domain specific definitions can be found in various literature sources. In the enterprise systems context for example, interoperability is *"the capability of enterprise information systems to collaborate in such a fashion that eventually either their mutual goals become fulfilled or their cooperation is dissolved in a controllable manner in case of a fault. Interoperability problems can range from simple technological incompatibilities to conflicts between business strategies."*[RK09].

Another aspect that is important in the IoT is that of autonomous systems that can act on their own. [BD15] provide a fitting definition for interoperability of autonomous systems in this regard:

> *"The ability of connected, autonomous, loosely coupled and possibly heterogeneous systems to coexist, to interoperate and to exchange flows (data and services, material or energy) to and from other systems while continuing their own logic of operation preserving their autonomy."*[BD15]

For the creation of interoperable systems it is important to determine, what the optimal level of interoperability is, i.e. in what ways should the systems work together and in what ways they should not. For example, if two systems interoperate by using a subset of their functionality, they should restrict to this subset and not exchange data at will, since that could impact privacy and security of users [GP12]. However it is still unclear, how the optimal level of interoperability can be determined and be made sustainable in IoT. We also need to distinguish between interoperation and cooperation. While interoperability is a capability that is necessary for interoperation it is by no means sufficient to guarantee cooperation, however a necessary prerequisite [Mun02]. However, in this thesis we try to investigate systems that not only interoperate but are also able to cooperate, towards a common goal and based on agreements.

2.5.2 Platform interoperability

This work specifically focuses on the interoperability concepts for IoT platforms. Why is interoperability of IoT platforms actually important? Why does one not only consider interoperability between IoT devices or sensors? As already motivated in section 2.3, nowadays, IoT platforms perform a central role in the engineering of IoT systems. Popular platforms allow to connect an abundance of sensors and devices in order to integrate them into new IoT solutions. Considering the fact, that IoT solutions would be build directly through addressing individual devices/sensors is unrealistic due to scalability and maintenance reasons. Therefore, solving interoperability for IoT platforms in the end also solves the problem of interoperability between connected sensors as well, but on another abstraction level.

Since IoT platforms are a natural evolution from the popular Cloud Computing (CC) paradigm that arose during the last decade, it is interesting to observe how this closely related domain handles the problem of platform interoperability. Interestingly, in Cloud Computing (CC), platform interoperability between different CC platforms is currently experiencing the same attention as IoT platform interoperability, which hints that indeed this problem is not solved for generic Cloud platforms. Looking back at the section about IoT platforms, it is clear that the majority of IoT platforms are deployed in the Cloud since they were effectively developed with the help of Cloud Computing technologies for the IoT domain. In the context of CC, interoperability is conceived as the capability of public or private clouds, and other diverse systems within the enterprise to understand each other's application and service interfaces, configuration, forms of authentication and authorization, data formats etc. in order to cooperate and interoperate with each other [6]. In CC, different layers of interoperability can be distinguished: (i) Application interoperability, (ii) Service interoperability and (iii) Platform interoperability [Hoa16]. Application interoperability, addresses **application components**, regardless on which level (SaaS,PaaS,IaaS) they are deployed to collaborate across different platforms to either deliver existing or new functionality [Hoa16]. These applications components can be of any scope, i.e. a monolithic application or a service that is used as part of a distributed application [KGR16].

Service interoperability is the ability of using services across different platforms through a unified interface [Hoa16]. Service interoperability effectively merges with platform interoperability, if the platform components that are supposed to interoperate are deployed as services. PaaS

[6] http://www.cloud-council.org/deliverables/CSCC-Interoperability-and-Portability-for-Cloud-Computing-A-Guide.pdf), accessed on 21.12.2019

components help developers in the complete lifecycle of building and delivering applications. Platform interoperability is the ability of using these platform components to interoperate [Hoa16]. Examples for platform components of CC platforms are [Gmb16]:

- Host environment
- Middleware
- Assembly and Integration environment
- Asset management
- Discovery & Publishing
- Protocols for web service
- Identity management
- Management at service level
- Measurement & monitoring

Comparing this list to the common reference architecture for IoT platforms in section 2.3, it becomes obvious that all of these components can be found in IoT platforms as well. The components are deployed *as-a-service* which effectively means, that the components are only used and offered on a as-needed-basis, which is usually a way to reduce costs. The service exposes information that can be used by service consumers to determine if the service is appropriate for their needs [Muf09]. Although multiple IoT platform provides can offer the same services through their platform for IoT developers, the underlying implementation of the service can vary between platform providers. As we will later discover, this affects interoperability and dependability of the system during runtime.

2.5.3 Metrics for interoperability

Clearly, with regard to the range of available interoperability definitions it is difficult to address it from a technical viewpoint. Usually interoperability is described inside a layer-based architecture. Various studies have already provided an adequate overview on interoperability assessment,

specifically [LGP16][For08][FCGD07] and [Gué14]. [LGP16] also provides a classification of assessment methods alongside the four interoperability levels. According to the review of interoperability measurement models by [FCGD07], there are 14 different interoperability measurement models of which only one is completely quantitative. In the years after this report, however, additional models have been developed that provide qualitative as well as quantitative measurements as described later. Therefore, assessment of interoperability can be loosely divided into two classes: qualitative level assessment (section 2.5.3.1) and quantitative assessment (section 2.5.3.2).

2.5.3.1 Qualitative assessment

According to [SBC+15], interoperability can be conceptualized into four dimensions:

Technical Interoperability is typically associated with hardware/software components, systems, and platforms that enable machine-to-machine communication. *"This kind of interoperability is often centered on (communication) protocols and the infrastructure needed for those protocols to operate"* [GMLF18].

Syntactic interoperability is the ability to exchange data. This type is usually associated with the definition of common data formats such as XML or JSON.

Semantic interoperability builds on the syntactic layer and requires a common understanding of the underlying data between the two interacting parties. On this level, ontologies are used to achieve the desired common understanding.

Organizational interoperability as the highest level, is concerned with the *"ability of organizations (in the enterprise context) to effectively communicate and transfer meaningful data (information) despite the use of a variety of information systems over significantly different types of infrastructure"* [Gué14]. In some literature sources, the syntactic and semantic interoperability layers are combined as *"conceptual interoperability"* [LGP16]. This makes sense, since usually syntax and semantic are closely related.

Interoperability barriers are further elaborated in [BPGG11], who provide a conceptual overview of different technical interoperability solutions in complex distributed systems. One can distinguish between the following interoperability barriers: *"data"*, *"middleware"*, *"application"* and

"non-functional heterogeneity". While the data level concerns syntax and semantic barriers, the middleware layer concerns heterogeneous protocols, applications with different use of underlying middleware and APIs and finally non-functional heterogeneity with all other properties that are relevant for interoperability, e.g. reliance on message latency. The proposed solutions by [BPGG11] will be explained in more detail in the next chapter.

The most popular metrics with regards to assessing the level of interoperability from a qualitative level are the **LISI**, **LCIM** and **MMEI** metric [FCGD07].

LISI The LISI metric for measuring interoperability maturity was proposed in 1998 by the MITRE Corporation [FCGD07]. It contains 5 levels of maturity: Isolated, Connected, Functional, Domain, Enterprise. Interoperability between systems is analyzed by an interoperability matrix. The matrix is created manually based on the generic interoperability level of each system and the specific interoperability level system pair [Gué14]. LISI only covers the technical interoperability layer, not the organisational and conceptual aspects. Therefore LISI was extended by the LCIM metric[LGP16].

MMEI [Gué14] propose a maturity based measure that covers multiple interoperability aspects (Maturity model for Enterprise interoperability (MMEI)). It addresses the main concepts of existing interoperability maturity models and tries to integrate them into one coherent framework. It is based on the Enterprise Interoperability Framework (FEI) [CDE06]. However, the authors mention that this model is still under development and can serve as a basis for further research and development. It covers the following maturity levels for enterprises: unprepared, defined, aligned, organized, adapted. For each level, they characterize the properties of interoperability concerns and barriers that are specific for this level.

LCIM The LCIM metric is one of the most cited metrics for interoperability in literature and was proposed in [FCGD07]. It is also the basis for the interoperability conceptualization in IoT ecosystems in this thesis. It defines six levels of interoperability: no interoperability, technical, syntax, semantic, pragmatic, dynamic and conceptual. These levels address different focus areas, where syntax and semantic deal with the format and content of information exchanged, pragmatic interoperability deals with the context of information, dynamic interoperability with the effect of exchanged information and lastly conceptual interoperability with a documented con-

ceptual model [TC09]. The model was originally developed for the Modeling&Simulation domain, however by now it is also used by other communities [TC09]. LCIM has a descriptive role that defines how interoperability looks like on each level and a prescriptive role that serves as a requirement set in order to reach the respective levels [TC09]. For example, in order to reach pragmatic interoperability, a method for sharing meaning of terms and methods for anticipating context are required [TC09]. With regards to the interoperability concerns, LCIM however only considers the data aspect [Gué14].

The LCIM metric is used as the baseline metric in this thesis due to its descriptive and prescriptive role. Also, it has been developed for the software design process [NDBC16], which is important to connect the metric to existing work in IoT interoperability and especially work on semantic interoperability. At the same time it allows to highlight the deficits with existing interoperability solutions with regards to the levels **above** semantic interoperability. It is however unclear how this framework connects to the runtime aspect of interoperability [NDBC16]. Hence, we will look in more detail into the background of the higher levels of interoperability in the LCIM metric (semantic,pragmatic) since they are the focus of attention during the rest of this thesis.

2.5.3.2 Semantic interoperability

Semantic interoperability describes a common understanding about the meaning of content that is shared between a group of entities [SBC+15]. According to the definition in the LCIM model, semantic interoperability is reached when *"the meaning of the data is shared; the content of the information exchange requests are un-ambiguously defined"* [TDT07]. Semantic web technologies play a crucial to reach semantic interoperability. Semantic web technologies originate from the Semantic Web movement in the 2000s where semantic web agents were supposed to understand the content of web pages. The developed technologies are now used in the IoT context where they should lead to *"interoperability among IoT resources, information models, data providers and consumers, and facilitate effective data access and integration, resource discovery, semantic reasoning, and knowledge extraction"* [Ini15]. Reaching a common meaning of data is usually achieved with the help of ontologies. An ontology describes the knowledge about a certain domain by dividing it into relations between these concepts. They allow information exchange between systems on different levels of abstraction which is vital in the IoT context [SBC+15]. Also, they play an important role throughout the whole semantic interoperability stack, from annotation to managing access and resource discovery [GPP+16]. An ontology is used by annotating terms (e.g. data items)

with semantic knowledge, thus making the data interoperable between multiple systems (assuming a common understanding of the underlying ontology). A formal introduction to ontologies is given by [DMA13]. There are a large number of ontologies developed already for the IoT space. An overview about popular ontologies from different domains is given in [SBC+15] and [GPP+16]. However, most of the IoT ontologies are still in prototypical stage, since they originate from the research domain [GPP+16]. An exception is the SSN ontology [7], representing the most important core vocabulary for sensing data. It defines the notion of sensor and physical devices in general, therefore formally the concept of a virtual sensor as a subclass of the sensor concept as defined in the SSN ontology [SBC+15].

Different specifications have been developed in order to create a formal language for the specification of ontologies. **RDF** - Resource description framework is a generic syntax in order to display data data on the Web. It is recommended by the W3C as a language to describe resources. Information in RDF is represented as triples in the form of *subject*, *predicate* and *object* or directed graphs, so called *RDF-graphs*. In order to retrieve information from RDF-based ontologies, the query language **SPARQL** [8] can be used which resembles the popular SQL database query language. The Web Ontology Language (**OWL**) is a family of knowledge representation languages for creating and publishing ontologies. It is about describing domain terms in a formal way so that software agents can understand them. A widely used serialization of ontologies is **JSON-LD**, which is an extension of JSON and RDF specifically suited to serialize ontologies to make them interoperable usable.

2.5.3.3 Pragmatic interoperability

In abstract terms, pragmatic interoperability ensures that a message intention and the effects of the message exchange between two systems is understood. Pragmatic interoperability is achieved when the **use** of the data is known to the inter-operating systems. The underlying theoretical model of pragmatic interoperability can be rooted (similarly to semantic interoperability) in the linguistic traits of pragmatics. Pragmatics is a subfield of linguistics and semiotics that studies the ways in which context contributes to / or changes meaning. In pragmatics as opposed to semantics, the meaning of an utterance is not objectively defined but created during the conversation in context. Pragmatics is about non-literal meaning - it concerns with what is intended, even when

[7] https://www.w3.org/TR/vocab-ssn/, accessed on 21.12.2019
[8] http://w3.org/TR/rdf-sparql-query/, accessed on 21.12.2019

no explicit intention is stated. Pragmatics is more concerned with the meanings that utterances in fact convey when they are used, or with intended speaker meaning. Context of an utterance consists of a speaker, the sentence which is uttered, the act performed in the uttering of a sentence and the hearer. The meaning of an utterance may include certain intended effect on the hearer, by for example implicit requests.

Moving from the linguistic to the computational world, the context of pragmatic considerations changes. For example, in [RK09] pragmatic interoperability is defined as achieved *"if the intentions, business rules, and organizational policies of collaborating parties are compatible with each other"* [RK09]. [TDT07] argues that for pragmatic interoperability, sender/receiver pairs are "aligned" about the expectations of the message exchange. Systems share an understanding of the context of the data exchanged and the associated information exchange can be characterized as data exchange for the right purpose. There may be implicit expectations about what content is sent/received and in what order. [Asu10] identified the following four aspects which need to be considered in pragmatic interoperability:

- Message exchange - the pre-requisite for any communication
- Message intention - the desired possible state of the world which a message sender can achieve through collaboration with a receiver
- Message use - how a receiver interprets the intention of the communicated information on message receipt - how the receiver acts upon the message to realize the intention of the sender
- Message context - the importance of context in the use, interpretation and understanding of the message is regarded by most authors as a core concept in pragmatic interoperability. Context helps in the correct interpretation of the meaning of the message request so that only the relevant information or relevant actions are used to accomplish the sender's intention [ABI+11]

These aspects add the importance of the user context in achieving pragmatic interoperability which is a topic that will be referred to later when discussing the runtime interoperability problem. Context information can consist of information about the system itself, goal, purpose, theme, time, location, etc. [LLL14]. Such kinds of context-awareness in processing data is what

is currently missing from IoT middleware solutions [RMjP15]. Pragmatic interoperability generally aims to meet users expectations in a collaborative processes [NDBC16], although it is to be defined who these users are. Another distinction can be made between the system and business levels. While on the system level, "pragmatic interoperability deals with sharing the same understanding of the intended and actual use of exchanged system message in a given context" [Asu10], on the business level it "goes beyond service use, by considering also the compatibility of business intentions, business rules, organizational policies, and the establishment and maintenance of trust and reputation mechanisms between collaborating business parties" [Asu10].
To summarize, we can understand the difference between semantic and pragmatic interoperability as such:

- Semantic interoperability deals with the **objective** meaning of data
- Pragmatic interoperability deals with the **subjective** meaning of data, in other terms a mutual agreement on the use of the data

2.5.3.4 Quantitative assessment

A measurement metric serves an important guideline for interoperability related decisions. Quantifying interoperability is an important concept that is still not fully explored and only few solutions exist. Compared to the qualitative approaches, quantification can be considered a detailed measurement of the underlying interoperability between systems. Here also the difficulty for designing such a metric lies. Even within the same domain different parameters are more or less important for the interoperability between systems. There are no uniform requirements even within domains. The key to measuring interoperability of systems lies in determining the basis of the measurement [FGCJ08]. In [FCGD07], the authors propose the "i-score"-metric which they later improve in [FGCJ08]. It is a mathematical method for measuring interoperability between various kinds of systems. This means, it was not specifically designed to be used by a particular domain such as enterprise, military or IoT. It is computed based on a so called "operational thread" [FCGD07], i.e. the sequence of interoperation between different systems. Furthermore it can compute the maximum possible interoperability and determine ways to close the interoperability gap between the current and the optimum level [FCGD07]. The advantage of this method is its entirely mathematical background, making it possible for optimization criteria to use this metric at runtime. [Nay15] create the *Interoperability index model* which is an improvement of the "i-score"-

metric by [FCGD07]. It improves the i-score in three areas: (i) They consider direct and indirect interfaces, (ii) they consider weights for interfaces, (iii) they define the *interoperability index* that is related with how the interoperability between systems in the operational thread is calculated. [JWDM13] developed a method that measures interoperability based on a reliability measurement which works at the system pair level and also at the system of systems level. The interoperability measurement is intended to be applied at the conceptual design phase to analyze interoperability between different system architectures. They analyze other measurement approaches and compare their approach with the "i-score" method by [FCGD07] to conclude, that the "i-score" does not measure the quality of interoperation but only the compatibility. [YAP12] focus on the conceptual layer and propose a method to quantify the semantic gap between conceptual models of Cooperative Information Systems.[JB12] present an approach for interoperability evaluation based on causal performance measurement models to graphically represent and measure the interoperability. Finally, another way to measure interoperability using Petri net models is given by [GW12]. This overview shows that indeed, measuring interoperability is a difficult and domain-subjective problem, where solutions are not easily portable. While the qualitative level is still rather general, the quantitative level relies on specific points of measurements which might not be available in other domains.

2.5.4 Interoperability - temporal axis

We can distinct interoperability on the temporal axis between design-time and runtime interoperability. [Wei14] consider the interoperability dimension of **"interoperability solution timing"**, that distinguishes between *"a-priori solutions"* and *"a-posteriori solutions"* to interoperability. Solution timings determine **when** the interoperability problem should be addressed. A-priori solutions comprise approaches that try to anticipate problems and to overcome barriers **before** systems are build, i.e. design-time approaches. A-posteriori solutions on the other hand are approaches that allow to identify and correct problems at runtime and thus support the formation and robustness of dynamic systems. In the context of this thesis, this fourth dimension is particularly relevant as discussed in later chapters. A-priori interoperability requires the interacting systems to agree up front on a particular level of interoperability. Usually, these agreements are only applicable to closed systems and exhibit rigidness since the interoperability requirements are fixed at *design-time*. An example of this type of interoperability would be an approach solely based on standards that defines the valid ways systems are supposed/allowed to interoperate. Systems that adhere

to the standard are *a priori interoperable*. This process requires a lot of foresight by the standard bodies and developers of the systems. Hence it is generally considered a quite slow design-time process. A-posteriori interoperability on the other hand involves continually adapting systems (components) to the current operational context. This is especially important in dynamic environments where it is unclear at design-time, which systems will actually work together. As we will see in the next section, the IoT generally constitute such dynamic environments, hence we will specifically concentrate on *a posteriori* solutions.

In systems theory, *a posteriori* solutions can be classified into three divisions: (i) exclusion, (ii) domination or (iii) adjustment. Adjustment is the most desired solutions since it places the least amount of restrictions on the systems operation, however it requires systems to be adaptable. Exclusion works as a protective mechanism, so that systems can be excluded from a collaboration when they show defective behaviour. Domination limits systems in their abilities which can help to prevent interoperability problems by restricting systems to a subset of functionality. All these approaches are classified under the umbrella term *coordination*[NGC09].

2.6 Summary

This chapter presented an overview about the two main topics of this thesis: IoT platforms and interoperability. The goal in the next chapters will be to connect the two concepts, i.e. deriving solutions for interoperability of IoT platforms. Therefore, in the next chapter we will review existing work on interoperability between IoT platforms before a practical approach to this problem is presented in the following chapters.

Chapter 3

Related work

3.1 Aim

We will review the state of related work in interoperability conceptualizations and solutions and will analyze how the existing methods relate to interoperability between IoT platforms. This serves as a state-of-the art analysis in order to evaluate why existing solutions have not yet solved the problem of interoperability between IoT platforms. In general we can distinguish between two broad classes of intertwined approaches: (i) standardization activities and (ii) software based solutions.

3.2 Standardization bodies for IoT interoperability

Standardization has always been the driving force behind interoperability and is considered the "go-to method" to achieve interoperability in distributed systems [Gas15]. Standards can be characterized as a collaborative approach towards higher levels of interoperability by multiple parties and can be distinguished as either open or closed which affects how accessible and widespread the standards can be used [Gas15]. If two systems adhere to the same standards, they are guaranteed to work together to the extent of the standard. Nonetheless in the IoT, standardization is not as easily achieved as in other domains which can be seen by the already large quantity of standards developed that lead to the problem of creating methods to bridge standards. Efforts on standardization have been spent by standard developing bodies, such as industry alliances, special interest groups and open communities [Gmb16]. However, their solutions are usually bound to specific areas or domains.

Standards and reference architectures are being created on several layers of the complete IoT technology stack, mainly on the communication, application and service layer [Gmb16]. While the communication level deals with accommodating communication standards for the IoT, the application level deals with device management and protocols (IEEE,Bluetooth SIG,...) and on the service level we find models and reference architectures on how to design IoT systems. This is possible since many core-level IoT services remain similar across verticals and solutions, so that it makes sense to standardize them although at the same time, a lot of IoT solutions rely on customizations which contradict this motion. Popular examples for service level standardizations are the AllSeen Alliance, Industrial Internet Consortium (IIC), oneM2M, OCF and FiWare's NGSI [Gmb16].

These working groups develop best practices and frameworks for the IoT space and consists of consortia of academic and industrial partners such as Microsoft, LG, Sony, among others [Gmb16]. Despite these efforts, a lot of challenges remain that concern privacy and confidentiality of exchanged data, standardized data exchange formats and interfaces within and across industry domains which are not yet solved by existing standards [Gmb16]. It is doubtful that there will be a universal standard for IoT that will cover all topics, but rather there will be a selection of standards and ecosystems that will become dominant in various segments [Gmb16]. Thus, the IoT has seen increased efforts trying to bridge standards.

Standardization is a double-edged sword in the IoT because, on the one hand, if all parties agree to standards, they provide a solid basis for interoperability. On the other hand, the standardization process is usually time consuming, complex and costly [Gas15] which is a particular problem in the open and fast paced IoT environment. Since the speed in which IoT technology develops is increasing rapidly, standards based on outdated assumptions might restrict future developments due to historical limitations [Gas15]. Standards specifications have to anticipate all possible future scenarios which is, to say the least, *very* difficult inside highly dynamic environments. As already proposed by [CFMP05], interoperability becomes a runtime problem which is especially true for IoT.

In summary, the standardization model in the IoT fails due to the following reasons:

1. Scalability: The number of IoT systems and platforms rises rapidly [IoT16] and efforts needed to adhere to new standards is not scalable due to development and synchronization efforts. Thus, the IoT standard landscape currently follows the same fait as previous standardization efforts from other domains - More and more standards are developed which are not

interoperable amongst another [IoT]

2. Requirements: Since in the IoT, systems are interconnected in various ways, it is difficult to foresee from a standard-development standpoint, how the systems will work together and thus, what capabilities the standard needs to have.

3. Time: Standardization is usually a long and time-consuming process. This approach does not work in the short-lived, rapidly evolving space of the IoT.

3.3 Software based approaches for IoT platform interoperability

Software-based approaches for IoT platform interoperability promise to offer more flexible and dynamic solutions compared to standards. Although there are a plethora of solutions, the most common solutions in the IoT are middleware based solutions. This is because middleware generally acts as an abstraction layer towards heterogeneity on the platform or application level and provides a key set of layers in distributed architectures since the early 90s, originating from basic client/server models to service oriented architectures and new communication concepts such as publish-subscribe patterns or message queues [BST16]. But research on which type of middleware architecture is the most promising in the IoT is still ongoing due to the following, IoT related challenges on the *usual* middleware approach according to [BST16]:

- (i) massive scale of IoT devices results in dependability, discovery and stability issues
- (ii) complex event processing and system architectures
- (iii) heterogeneity of IoT systems
- and (iv) privacy, trust and safety for accessing IoT data.

Traditional middleware approaches achieve interoperability by relying on all systems to be implemented on the same middleware architecture. Each peer uses the same middleware layer for communication, so that interoperability can be guaranteed. This is essentially equivalent to people agreeing on speaking a common language. Common examples for such communication layers are: RPC based systems (e.g. CORBA [VF13]) , message-based, publish-subscribe and tuple spaces [BPGG11]. A disadvantage of this approach in dynamic environments is the requirement that every application needs to implement the same middleware [BPGG11]. Another approach is

the use of interoperability platforms, that follow a translation approach guaranteeing that applications can interoperate with all services irrespective of the employed middleware architecture. The platform provides an API for developers and a substitution mechanism to translate between disparate protocols between middlewares. Software bridges are third party infrastructure that facilitate translation between different legacy middleware and *"act as a one-to-one mapping between domains; it will take messages from a client in one format and then marshal this to the format of the server middleware; the response is then mapped to the original message format"* [BPGG11]. A particular example of a software bridge is the Enterprise service bus (ESB) which offers a N-1-M mapping between different messaging systems. Yet another approach (logical mobility) relies on clients downloading code fragments for integration to be interoperable with other systems [BPGG11]. This encapsulates any implementation details of other services and spares the developer of the client application to implement them himself. Instead, the downloaded code will be used as is to handle all interoperability related functionality. The downside is, that there needs to be a common platform to run the code on (e.g. a common agent platform) which weakens the interoperability potential [BPGG11].

Overview over IoT middleware landscape The landscape of IoT middleware is as diverse as the IoT platform landscape with a recent comprehensive overview given by [RMjP15]. They show that IoT-middleware is usually applied on two different levels: middleware hosted in the cloud (commonly refereed to as IoT platform) and middleware that is using a backbone network. Inside these levels there has been a plethora of middleware solutions, developed using different design approaches [RMjP15]. The general distinction is between event-based, service-oriented, VM-based, agent-based, tuple-based, database-oriented and application specific middleware solutions [RMjP15], with hybrids among these approaches. Service-oriented approaches have become the most popular approach, hence they are focused in later sections. The reason is that the IoT does not only concern data that is exposed to the outside, but also resources such as sensors, actuators or entire networks. Recently, a trend can be observed to abstract IoT resources as services, where services provide an interface towards the resources, acting as virtual counterparts of physical devices. This achieves a common communication layer that can abstract device heterogeneity and also join with other data sources easily. By implementing the service interface, resources provide and consume capabilities to the outside [Del13]. Services in this context can be formally defined as: *"coarse-grained, discoverable, and self-contained software entities that interact with applications and*

other services through a loosely coupled, often asynchronous, message-based communication model" [CFMP05]. This view merges the Internet-of-Things with the *Future Internet* vision of the Internet-of-Services and porting existing technologies that were originally developed for the service domain to be used in IoT.

The following list contains some of the more prominent examples of IoT middleware:

- **UBIWARE**: UBIWARE is an agent-based middleware where each resource is modeled as a software agent. The agent monitors the conditions of the resource and enables the resource to interact with other elements. In general, UBIWARE allows the automated discovery, orchestration, choreography, invocation and execution of IoT resources [Bat13].
- **HYDRA**: HYDRA is a service-oriented architecture which uses web service for the discovery and description of IoT resources based on XML and web protocol standards [RMjP15].
- **OpenIoT**: OpenIoT is an open-source middleware which follows the idea of on demand access to IoT services which are offered as cloud services. OpenIoT refers to this as the 'Cloud-of-things' [Bat13]. OpenIoT resources are addressed through URIs as OpenIoT adheres to the REST paradigm. Interactions with the IoT resources are then offered as RESTful web services [Bat13].
- **Hypercat**: Hypercat provides a hub-based approach to solve IoT interoperability through an IoT catalogue specification [BL15]. They describe a mechanism to adapt an existing IoT platform to use the HyperCat specification. The IoT platform thus becomes a hub that exposes *Smart objects* through a well-defined RESTful API, and a common hypermedia specification (HyperCat). HyperCat is an outlier among the other frameworks as it only focuses on discovery. This functionality, however, is solved in a very lean and lightweight manner.

3.3.1 Service oriented middleware

The most frequently used type of middleware architecture in the IoT space is the Service oriented architecture (SOA) originating from the service oriented computing (SOC) domain [AIM10]. Service-Oriented Computing (SOC) is a computing paradigm that utilizes services as fundamental elements to support rapid, low-cost development of distributed applications in heterogeneous environments [PG03]. The SOA architecture makes it possible to decompose monolithic applications

into fine-grained, independent services that do not possess any references to other services which provides dynamic business processes and more agile applications even across organizations and computing platforms that can adapt at runtime [PG03]. Interoperability is achieved through SOC by abstracting specific characteristics of each individual system through a common service layer [PG03].

In SOA and web services, two roles are differentiated: a service requester (client) and a service provider, which communicate via service requests. A role thus reflects a type of participant in an SOA. The design principles of an SOA are independent of any specific technology, such as Web Services or J2EE Enterprise Java Beans. In particular, any technology that complies with WSDL and communicates with XML messages works with SOA, however web services have become the preferred implementation technology for realizing SOAs. SOA have been used in the past successfully to develop interoperating web-service architectures [Muf09]. Key factors that SOA contribute to enhanced interoperability are: standard interfaces, dynamic binding at runtime, loose coupling, dynamic service contracts & flexibility.

Recent IoT middleware architectures often follow the Service Oriented Architecture (SOA) approach [AIM10], notably *Hydra*, *MUSIC*, *SenseWrap* among others. However their focus on how to enable interoperability differs, with some approaches abstracting devices while others abstract data/information[THIG11]. An IoT middleware needs to fulfill various requirements, among other things the discovery and management of IoT resources and devices, data and events while achieving scalability, reliability and availability [RMjP15]. On an architectural level, an IoT middleware needs to provide interoperability through programming abstractions, such as service-oriented technologies, be service-based and provide distributed context-awareness [RMjP15]. A common point of agreement is the use of semantics and metadata to overcome heterogeneity issues [THIG11].

Another interoperability approach which has been becoming more and more popular, especially in the early 2010s during the Cloud Computing boom, is the **Service brokering platforms**. A service broker is a specific type of middleware, that mediates between service consumers and producers by offering search, aggregation, integration and security for customers of cloud services. Naturally this fits the introduction of service oriented architecture in which everything is offered and provided as services. The advantage of the service broker is that the consumer has easier access to a broader rage of services compared to manual search. The service broker ad-

dresses interoperability by reducing the complexity of integration of other services and reducing operational problems. But the service broker model also features a couple of disadvantages such as: (i) the reliance on a centralized entity in order to find and aggregate services and (ii) fixed roles of the involved actors. Therefore, alternative solutions to the broker model should be considered as well.

3.3.2 Semantic interoperability solutions

Semantic interoperability is recognized within the IoT domain as an important problem to solve due to the inherent heterogeneity of communication protocols, standards and methods[SBC+15]. The view of a "Semantic oriented" IoT vision has thus emerged in recent years, with a plethora of solutions addressing semantic interoperability. It is a common point of agreement that semantic technologies provide the necessary foundation for interoperability in the IoT and at the time of writing, there are a lot of research efforts dedicated to semantic interoperability in order to improve the exchange of machine interpretable information between IoT devices and platforms.
Semantic technologies have experienced extensive research to achieve web service interoperability in the early 2000s (Semantic web services). The goal of semantic web services are self-describing services with rich semantic capability descriptions that are automatically discoverable and composable. Approaches for semantic interoperability are closely linked to the Semantic Web activities, which serves as a framework to achieve semantic interoperability. Semantic Web technologies such as the Resource description framework (RDF), RDF- Schema and the Web Ontology Language(OWL) are increasingly used within IoT [SBC+15]. The Semantic Web community also established the ideas of *Linked Data* for structuring data so that it can be interlinked[SBC+15], as well as OWL-S, a powerful set of ontologies to describe and reason over service descriptions *"through a representation of the semantics of the operations and the messages of the service"* [BPGG11]. Middleware architectures and service brokers which are utilizing semantic technologies for data exchange and discovery are called *semantic middleware*. Semantic middleware creates a common framework that enables data sharing and exchange across distributed devices, applications and locations. In this way, semantic middleware establish semantic interoperability. Semantic interoperability is not only important for describing data and metadata but also for the different IoT components or services to be discoverable, accessed and manageable. Semantic interoperability will establish the *"effective discovery, query, interpretation and integration of the IoT data"* [VF13].

[SBC+15] provide a detailed overview about semantic interoperability challenges and solutions and give hints how to solve these by providing an extensive research roadmap. They describe semantics as the starting point of semantic interoperability, with future research needed on semantic reasoning and interpretation on the data for automated processing. Since these techniques are notoriously challenging to realize on constrained devices and platforms, there are approaches needed in order to improve the performance and resource consumption of these concepts. Ontologies are a solid basis for semantic data exchange but ontology encoding is still quite time consuming and current solutions are not applicable for real-time constraints [BI15]. Ontology matching solutions appear as a promising solution to overcome ontology heterogeneities but they are more complex than pure ontology encoding, therefore not yet widely available. [VF13] list among the essential challenges for semantic interoperability the integration of multiple interconnected smart objects (sensors & actuators), automated linking of relevant data sources (annotation), creation and management of virtual smart objects, discovering smart objects, data sources and analysis and reasoning.

Technical and semantic interoperability has also received much attention among European research projects and also standardization committees such as the W3C WoT [1]. These projects can be classified into two groups which either work on architectures or communication protocols to solve semantic interoperability [Bor14]. An overview of research achievements by 12 EU projects given by [SBC+15] describes in detail the contribution of each project to the challenge of IoT semantic interoperability. Comparing the approaches, only OpenIoT [2] has addressed all requirements for semantic interoperability that were described earlier in [SBC+15].

3.3.3 Solutions to address pragmatic interoperability

Besides semantic interoperability, pragmatic and dynamic interoperability also play an important role in the IoT space since it aims to meet users expectations in the collaborative processes. As outlined in the previous chapter, pragmatic interoperability is achieved when the **use** of the data is understood by the interoperating systems [AV11], i.e. the context of the application of data is clearly specified. Pragmatic interoperability builds on top of semantic interoperability and can be

[1] https://www.w3.org/WoT/, accessed 24.12.2019
[2] http://www.openiot.eu/, accessed 24.12.2019

considered a more sophisticated way for services to choose other service in SOA [Muf09]. However, as discussed in [Muf09] and [NDBC16], there is still little work done to technically realize pragmatic interoperability in addition to a lack of experimental studies. At the time of writing there are currently no solutions yet that specifically deal with pragmatic interoperability in the IoT on an implementation ready basis. Service discovery, composition and/or selection and ontologies are among the most common tools to achieve pragmatic interoperability as of now[NDBC16]. Furthermore, the most frequent application domain is e-business with currently no existing approaches for the IoT. Ontologies and metamodels are used to represent pragmatic knowledge which is further used by service brokers to find appropriate (web)-services that match user expectations. [Liu07] propose an evolution from the semantic web to pragmatic web. They postulate that pragmatics, concerned with the use of information in relation to the context and intended purposes, is extremely important in web service and applications. In a pragmatic web, the web services need to be able to facilitate communication and negotiate between service consumers and service providers. After reviewing the current work in pragmatic web, the paper presents a semiotic approach to website services, particularly on request decomposition and service aggregation and present a new design for pragmatic webservices. The pragmatic web covers the semantic web drawbacks of missing the user intentions and intended purposes and effects of communication and put ontologies into context to deal with partial, contradicting ontologies. This makes it a relevant related domain to be considered in an IoT context! [TE06] provide a semantic resolution of web service requests. Even if potentially matching webservices, are matched based on semantic knowledge they might still not be compatible if used in a different situational context. Collaborators therefore have to continuously communicate and share background/context knowledge. During the web service discovery, they propose a pragmatic methodology which captures the context of the web services usage. According to this methodology, the context of web services' usage can be determined by considering the context of collaborators themselves. They argue that the context of collaborators must be well defined at the time of offer/demand definition for web services. To model such a context they propose the exchange of standard XML-based message types that are to be interpreted identically by all collaborators.

3.4 Summary and gaps in current interoperability research

IoT ecosystems are especially challenging for interoperability since they are complex systems, due to the massive scale of devices, complex event processing and heterogeneous system architectures. There is a general consensus in the analyzed literature, that semantic technologies and service-oriented based middleware are a favorable approach. However, the literature also points to the abundance of existing standards and middleware solutions which have not yet sufficiently solved all aspects of the interoperability problem. Furthermore, IoT interoperability solutions are usually domain specific. If a domain-independent, universal interoperability solution would be possible is still an open question. Dynamic solutions (possibly machine learning supported) are still not used in production and only developed in research prototypes. The promise of such dynamic interoperability solutions would be a reduction in complexity due to the increased automation.

Achieving semantic interoperability in the IoT is a multi-step process and still automatic solutions are missing for parts of the process [GPP+16]. Also, most of the analyzed service binding methods in current SOA architectures are performed at design-time which does not yield the full potential of SOA [Muf09]. Ontology mismatches between and within domains are the main reason for this. This is especially problematic, when we consider the goal for dynamic interoperability solutions in IoT.

Yet, semantic interoperability is only one part of realizing true interoperability in the future Internet of Things. There are still no interoperability projects addressing higher levels of interoperability such as pragmatic interoperability. This is likely because the difficulty of interoperability increases with the considered level [Muf09] and clear requirements for pragmatic interoperability are still missing [NDBC16]. It is also still an open issue, how to map the business/process and organizational levels of interoperability to IoT. The framework provided by [DM14] from enterprise systems engineering is a good starting point, as it already covers the three dimensional interoperability problem space. However, this framework is still bound to design-time analysis. Additionally, interoperability assessment is still an open issue in future enterprise systems, which makes it also a problem for IoT ecosystems [PZJGR16] .

There is generally no approach to consider interoperability from a formalized perspective, as

proposed in [NLGC10]. Although the qualitative partitioning into layers makes interoperability more accessible, it is still vague what system requirements are derived from this. Also a more fine granular distinction in between the classes of interoperability might be necessary for the IoT. According to the work by [LGP16], the MMEI framework is the only method that sufficiently covers all conceptual, technical and organizational interoperability aspects. However the analysis, as with all the other maturity models is performed at design-time, before the actual interoperation starts. This makes the metrics not directly applicable for the dynamic interoperability assessment for IoT systems at runtime. Since MMEI measures interoperability before partners are known, while other methods only measure interoperability after the partners are known but only for specific aspects, there still exists a gap that needs to be filled [LGP16].

To summarize, the main weaknesses of existing interoperability solutions are:

- The main focus is on design-time approaches
- Existing solutions are mainly domain specific
- Higher interoperability levels have not yet been considered
- Interoperability formalization & assessment missing

The remainder of this thesis addresses these open issues to close the interoperability gap between IoT platforms and to develop solutions that provide reliable interoperability in IoT ecosystems.

Part II

Building the interoperability model

Chapter 4

A conceptual model of IoT ecosystems

4.1 Aim

How interoperability between different heterogeneous IoT platforms operated by different organizations, needs to be designed so that it leads to a working IoT ecosystem is an open issue. To address this issue first of all requires a proper concept for what an IoT ecosystem actually is. To conceptualize an open IoT ecosystem requires first to take a deeper look at the individual elements and components of such a system. After the content of an IoT ecosystem is understood, the next hurdle is to actually model it so we can evaluate theories and implementations through it. The first section 4.2 wdill introduce the reasoning behind the theoretical concepts to model IoT ecosystems, in particular systems-of-systems and digital service ecosystems theory. Section 4.3 will then present a conceptual model of cross-platform IoT ecosystem using these theoretical concepts. The model will be validated using existing literature from the IoT domain. The final section 4.3.6 then compares the properties of traditional centralized to decentralized IoT ecosystems and the implications on interoperability and explains why existing interoperability solutions are not sufficient in this model. The IoT ecosystem conceptualization will guide the development of interoperability concepts in the remainder of this thesis in the preceding chapter.

4.2 Modeling theory

4.2.1 Systems of systems

If the vision of the IoT will be successful that it will connect the physical and virtual world and that it will make knowledge widely reusable, IoT will inevitably become a giant system of systems.

Therefore, understanding the Internet of Things from a system of systems standpoint is not only an academic undertaking but has practical relevance as well. Many colossal challenges remain among which many relate to system-of-systems engineering. Systems of systems or complex systems theory are two (sometimes interchangeably used terms) to describe complex systems that are formed of independent constituent systems. In the context of IoT ecosystems, this theory becomes more and more relevant due to the tremendous growth in devices and platforms in the IoT which need to be managed.

> "System of systems research is about how to design, engineer, maintain and evolve a composition of subsystems while acknowledging the fact that these subsystems remain independent, are serving their own functions, and have their own management and lifecycles" [Luk16].

A System of systems (SoS) is more than a collection of subsystems, since the subsystems form their own connections and through diversity of the components and emergent behaviour new capabilities are established that fulfill rising demands inside a SoS [BS06]. Many of the established assumptions in classical system design, such as that the scope of the system is known, that the design phase of a system is terminated by an acceptance test or that faults are exceptional events, are not justified in a SoS environment[CBF+16] . The most influential assumptions that SoS engineering addresses are the system scope, the clearly bounded design-phase of the system and exceptions that occur during systems operation [CBF+16]. Also system boundaries, that separate the system from its environment are dynamic in SoS which makes it hardly possible to define a stable boundary. The term of Systems of systems has developed out of the emerging complexity in traditional systems engineering. It originally derived from the military domain, where a lot of complex military systems such as missile systems and command and control systems had been developed separately to fulfill a specific purpose. When the need arose to connect them with the goal of creating a *better* system that has capabilities, none of the single systems had, the term system of systems was created [SBV10].

The first formal definition for System of systems (SoS) was provided by [Mai98]. Fundamentally, SoS are structured around a collaborative rather than a directed structure and are not characterized by central management.

> *"A SoS is an integration of a finite number of constituent systems (CS) which are independent and operable, and which are networked together for a period of time to achieve a certain higher goals. The systems that are at the bottom of this hierarchy are called components"* [CBF+16].

To understand a SoS, first of all it is necessary to interpret the term *system*. [HF56] introduced one of the oldest definition of a system in a general sense:

> *"A system is a set of objects together with relationships between the objects and between their attributes, where objects are components of a system in unlimited variety, attributes are properties of objects and relationships tie the system together"* [HF56].

The environment of a system contains the entities and their actions that are not part of a system but have the ability to interact with the system. The system boundary marks a dividing line between two systems or between a system and its environment. A system is assembled to fulfill a *purpose*! The purpose, or sometimes called *objective* of a system is the systems goal at any given time. Oftentimes it is composed of sub-objectives that can change according to the situation [NGC09]. The goal of a system directly influences its thinking, structure and function [NGC09]. A paradigm that is usually addressed when discussing systems is the continual flow of input → throughput (processing) → output → feedback. The outputs directly affects the purpose of system, either positively or negatively. Another distinction can be made between *open* and *closed* systems. Closed systems do not interact with their environment. Since these systems are rather limited in functionality, they will not be addressed in this thesis since this would not be a reasonable assumption for IoT systems. In contrast, open systems receive information to interact dynamically with their environment.

A SoS is in itself a system and only the unique characteristics of SoS distinguish it from a *normal* system [Mai98]. As the constituent systems of a SoS are independently developed and operated, SoS can thus be considered as distributed systems. Consider for example an intelligent vehicle infrastructure that consists of vehicle-to-vehicle communication and vehicle-to-infrastructure communication as presented in [Luk16]. A vehicle itself would not constitute a system of system as it does not consist of systems that could be operated alone. But, a vehicle-to-vehicle communication scenario would in fact describe a system of system, since each vehicle is a system that possesses operational & managerial independence from its peers in a system-of-system. However, when

considering such communications, emergent behaviour can result from this interaction, that could easily be dangerous if not controlled. Also the geographically distribution of this system of system becomes apparent when considering all vehicles in a district, for example, that communicate among each other. Once we introduce vehicle-to-infrastructure communication, the complexity of the SoS increases dramatically, for instance when vehicles communicate among themselves **and** among the surrounding traffic lights in order to collaborate to provide a common goal that could shorter stopping times for all. But if the SoS is disassembled, the vehicles still drive and the traffic lights are still regulating traffic, i.e. the distinct systems still achieve their purpose.

Since SoS are applied in a variety of domains, this work restricts to **software intensive SoS**, since IoT systems are to a large extend software based. Software intensive SoS are a subclass of generic SoS, where software is a dominant factor in its composition [GCB+14].

SoS characteristics Due to the relative youth of SoS research, the area still suffers from a lack of appropriate formalism [DV15]. Generally, SoS are defined along two dimensions, the first one describing types of SoS while the second one describes typical characteristics of SoS [GCB+14] which are according to [DV15]:

- "Operational & Managerial independence of the constituent elements"
- "Evolutionary development"
- "Emergent behaviour"
- "Geographic distribution"
- "Coopetition" (cooperation of competitors for mutual benefits)[Coo]

4.2.2 Digital service ecosystems

> *"A Digital Ecosystem is a self-organizing, scalable and sustainable system composed of heterogeneous digital entities (species) and their interrelations focusing on interactions among entities to increase system utility, gain benefits, and promote information sharing, inner and inter cooperation and system innovation. The main characteristics of dig-*

ital ecosystems are : (i) self-organization, (ii) scalability, (iii) sustainability and (iv) dynamism" [LBB12]

Digital ecosystems are a manifestation and extension of the SoS concept and provide a solid scientific foundation for the generation of new hypotheses regarding delivery of value-added services inside IoT (cross-platform) ecosystems. The term digital ecosystem is inspired by analogies with natural ecosystems and was introduced in the mid-2000s. It has emerged due to software systems that grew more and more complex, and hence were increasingly difficult to manage [AO16]. Inspiration for digital ecosystems originates from biological ecosystems that can form tough and robust architectures hence the motivation to create dynamic/ robust and adaptable software infrastructures to design heterogeneous digital systems. An introduction to the characteristics of digital ecosystems is provided by [CW06]. Similar to the SoS introduction, a digital ecosystem can be divided into two key elements: species and an environment. *"The environment of a system is a set of elements and their relevant properties, which elements are not part of the system but a change in any of which can produce a change in the state of the system."*[Ack71]. To draw a clear distinction between the entities in the environment of the system and the entities within the system itself is usually not possible, since it depends on the point of view of the observer [Ack71]. The species interact with each other through the environment, which provides the *"infrastructure that supports the description, compositions, evolution, integration, sharing and distribution of its components"* [LBB12]. This infrastructure is provided by an *infrastructure provider* through the use of supporting services. It forms the foundational living environment for digital species but on the other hand digital species also enrich their environment by proving technologies to the environment [LBB12]. The environment of a digital ecosystem is realized by technologies which are, according to [LBB12], *extended web architecture*, *ontology based knowledge sharing*, *swarm intelligence architecture*. Ontology based knowledge sharing already refers to the concepts described in section 2.5: it is supposed to provide ontological concepts that are used to share knowledge between the digital species.

A digital service ecosystem (DSE) is a digital ecosystem where the species are modeled as services [IOKP16]. It is formally defined as *"being an open, loosely coupled, domain-clustered, demand-driven, self-organizing agents' environment, in which each species (human, economic species and digital species, i.e. computer, software and application) is proactive and responsive for its own benefit or profit"* [IOKP16]. A DSE allows a more fine grained distinction of its constituents into

the following elements: *"ecosystem members"* (service providers & consumers, service brokers and infrastructure providers), *"ecosystem capabilities"* and *"digital services"* which are produced/consumed by the ecosystem members [AO16]. The capabilities represent the properties of the ecosystem and how they are implemented by the ecosystem supporting services that are provided by the infrastructure provider. *"A digital service is any added-value that is delivered digitally"* [IOKP16]. It is automated entirely and ideally controlled by the customer of the service [IOKP16].

Each member inside a digital service ecosystem performs dual roles, i.e. they act both as clients and servers at the same time. This already shows the difference compared to traditional system architectures, where the roles of participating systems are fixed at design-time or integration time. A self-organizing system consists of multiple components that can change their interrelations. The self-organizing nature leads to different architectural models through group intelligence, where local interactions between agents determine the global behavior. As an example, [CW06] describe the formation of "hierarchy of swarms", "sequential workflow interaction" or "circular architectures". Collaboration between species is usually due to mutual interest, rather than forced/predetermined collaboration. Since digital ecosystems rely heavily on such collaborations, [CW06] specifically point to the evolution of digital ecosystem architecture and explain the differences to *classical* architectural patterns such as centralized (client-server), distributed (P2P) and loosely coupled (SOA). For example, unlike in web service based architecture with centralized brokers and distributed service requesters and providers, in digital ecosystems there is no such centralized control or fixed roles [CW06]. Collaboration in digital ecosystems relies on "swarm based behaviour", i.e. populations of simple agents that interact locally with one another and their environment. Essentially, a digital service ecosystem does not require a shared development platform for all members, compared to software ecosystems [IOKP16]. This opens the possibility to distribute the service ecosystem infrastructure.
Summarizing, digital service ecosystems provide a scientific and technological founded paradigm to design autonomic software systems for dynamic collaborations. However the concept has not yet properly been applied to the Internet of Things context which will thus be explained in the next section.

4.3 Conceptualization of IoT ecosystems from a DSE perspective

Applying the previously introduced theory allows to conceptualize IoT ecosystems from a digital service ecosystem perspective.This conceptualization is important in order to grasp the complexity of interoperability in the IoT, with regards to the open environment view, so that a proper interoperability conceptualization for cross-platform ecosystems can be derived. This helps to understand why existing approaches are not sufficient to achieve interoperability in such complex ecosystems. The conceptualization thus provides answers to the second research question of this thesis. For comprehension, exemplary use cases is introduced, partially originating from the use cases envisioned and evaluated in the European Horizon 2020 project BIG IoT project [IoT]. The conceptualization establishes three key elements: (i) the core system components, (ii) the roles& tasks inside the ecosystem and the interactions among the species and (iii) the properties of the IoT-DSE.

[MCG+14] present an overview about the current scenarios and approaches in the development of IoT-based SoS. They identified a couple of research challenges, since the combination of both worlds is still quite new and several open challenges remain. IoT-based SoS are highly heterogeneous in terms of physical devices and the surrounding systems which make up the ecosystem [MCG+14]. For instance, from a SoS standpoint, traditional software and systems engineering methods are inadequate for addressing the interoperation of the constituent systems of a SoS mainly due to their inherent emergent behavior and operational and managerial independence [MCG+14]. They conclude by highlighting the importance of middleware platforms to abstract away the heterogeneity and interoperability issues regarding physical devices, from the IoT perspective, and the heterogeneous constituent systems, from the SoS perspective thus combining the importance of platforms for the IoT and the SoS perspective. [BST16] discussed such middleware for System of systems (SoS), however they concluded that research in this domain is just starting to emerge. A number of European projects have also been established to investigate an IoT systems of systems perspective but the distributed systems view is not always fully represented in these projects.

4.3.1 Motivating examples

The following exemplary use cases showcase different compositions of IoT systems into complex IoT ecosystems and are used throughout this thesis to explain the theoretical concepts in a practical, real-world context. The use cases can be clustered into the following three domains: Agriculture, Industry & Smart city. These examples highlight the general trend in the Internet of Things towards service-oriented applications and integration of third-party services over the Internet. As a consequence, the capabilities and quality of service-oriented systems more and more will depend on the quality of its third-party services. Specifically, this means that service-oriented systems have to become resilient against failures of their third-party services and proper integration and interoperability has to be made sure.

4.3.1.1 Agriculture - Smart farming use case

More and more the digitalization reaches out of the smart cities and into the farming areas. Farmers more and more see the benefits of connecting the multitude of their machines in order to gain benefits in efficiency, cost or yield production. As this trend increases, the interoperability problem here becomes more and more apparent since usually a farm employs a large set of heterogeneous machines which are connected to different software backends for data collection and aggregation which means, for a fully connected farm management system, these various data sources need to be connected.

In this exemplary use case, a farm manager is interested in monitoring and doing task management with arable farming machines from different manufactures. The goal is to make the machines from different providers work seamlessly together with the rest of the farming equipment. The machines are equipted with sensors and actuators to sense the environment as well as to react to external control inputs sent over a wireless network. Work orders should be sent from the Farm Management Information System (FMIS) to the appropriate machines and after the tasks have been executed the work records should be send back to the FMIS. For the farm manager it is vital to send and retrieve data from his machines in real-time. The interoperability problem is thus a dual problem of actuation and sensing. The manager's intention is to optimize yield production as well as to reduce cost while a seed manufacturer uses the FMIS in order to gain information on how to optimize his production. The farm machine provider on the other hand operates the farming machine platform which acts as the data provider. The use case is illustrated in figure 4-1.

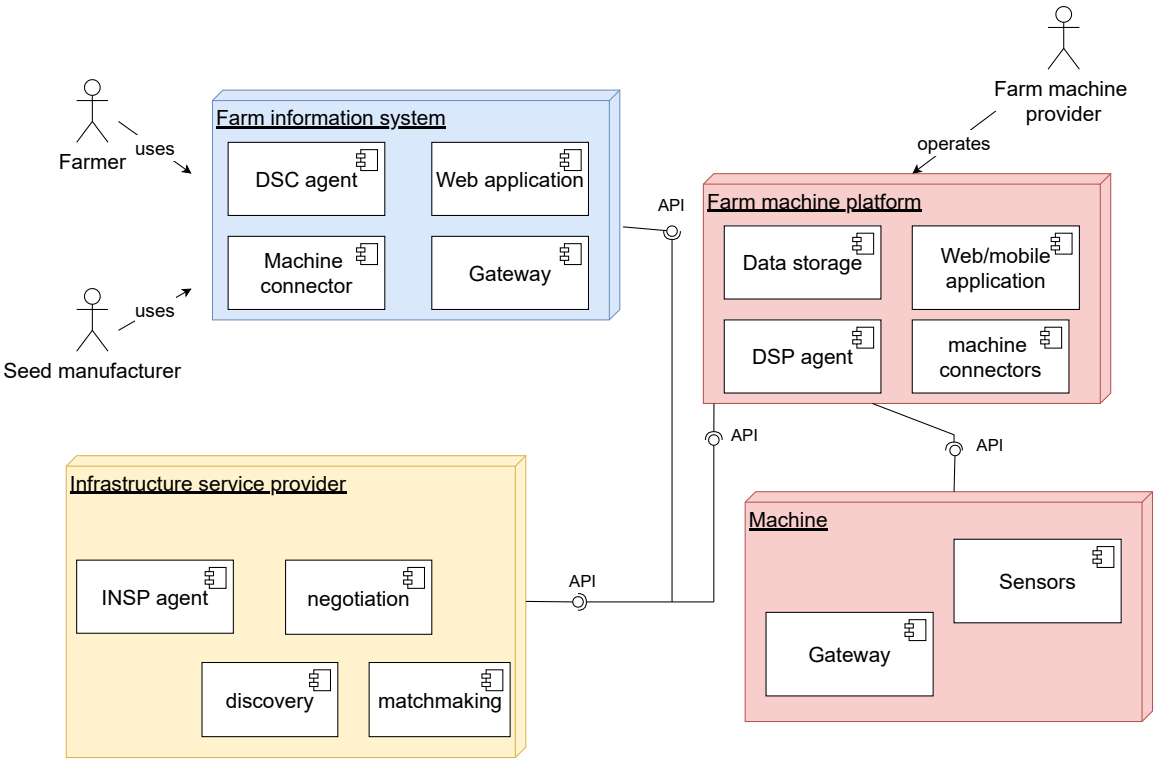


Figure 4-1: A smart farming use case, consisting of an infrastructure service provider, a farm information system, and a farming machine platform. The systems are used by three types of stakeholders (farmer, seed manufacturer and farm machine providers)

4.3.1.2 Industry - Smart production services use case

The vision of the Industry 4.0 concept lies in the assumption that data can be traded through a transaction platform in an automated and on-demand fashion [aH15]. Through trading machine data, production can be provided in a demand-driven way, custom-fit to specific production demands. This will add new revenue streams for machine manufacturers but also for third parties who contribute services to custom-fit production data. But, this vision has not yet become a reality mainly due to the problem of interoperability and missing marketplaces for trading this type of data. Specifically in the Industry 4.0 context, data is characterized by its multi-faceted nature, complexity and detailedness [aH15] which makes it difficult to design standardized solutions for trading production data. Also, a considerable number of industry standards already exist which are not interoperable. The interoperability focus in this use case lies on the correct and real-time provisioning of process parameters, since data has to be made available on demand quickly as well as custom-fit. The use case is illustrated in figure 4-2.

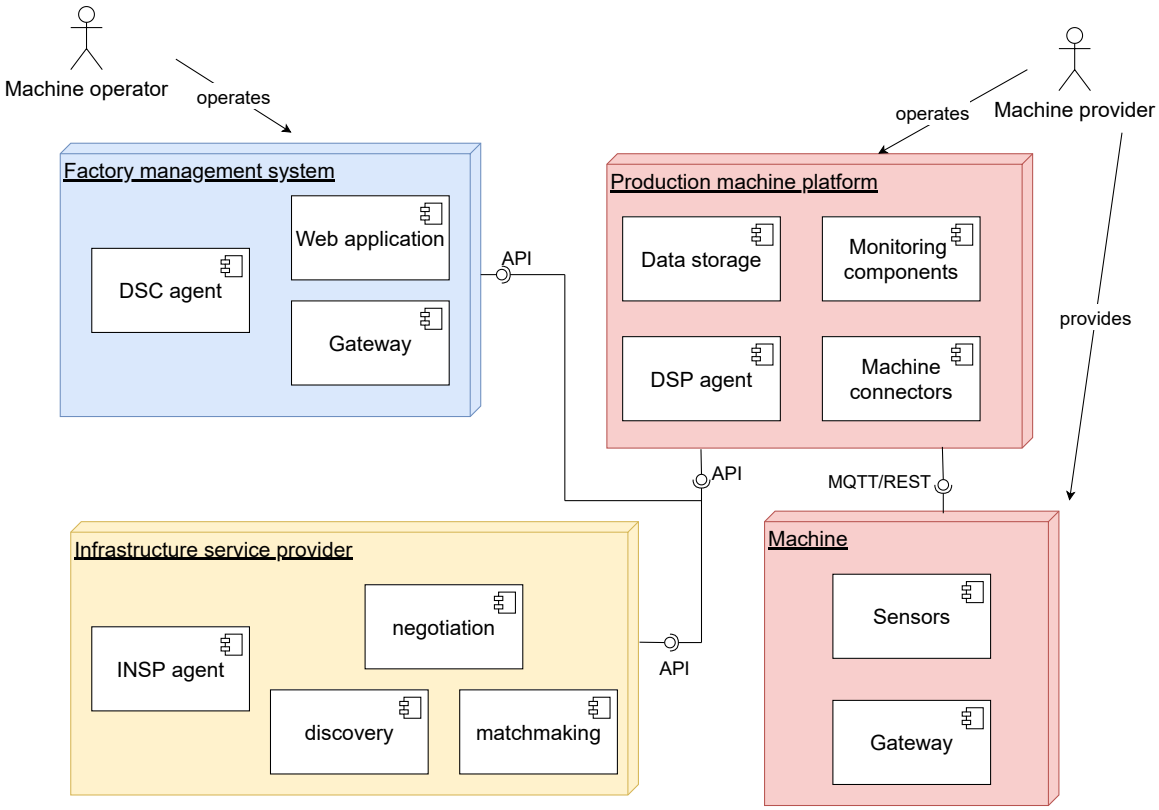


Figure 4-2: Smart production use case showing the involved systems and components, consisting of an infrastructure service provider, a factory management system and a production machine platform connected through an API.

4.3.2 Smart city - Environmental-aware routing service

The purpose of the environmental-aware routing service (4-3) is to deliver a service which, alongside a standard routing functionality known from car-navigation systems, includes additional real-time data sources about air quality and noise pollution which is especially relevant, for example, for cyclists. This information gets combined with other routing related information to supply an optimal route for a cyclist with the least amount of health impact. Such a use case is especially relevant in todays major cities which become ever more crowded with traffic and pollution. The IoT's value proposition is to combine physical services, such as real-time sensor-based evaluation of air quality with virtual services such as navigation and traffic information. The agglomeration is a significant value addition for customers, and reaches far beyond what is currently available in the service landscape of major platforms. To enable this service, interoperability is key. Opposed to the previous use cases, this one exemplifies a large open ecosystem of services and data which makes interoperability between these systems especially challenging. The essential elements with

regards to interoperability in this use case are the integration of the three IoT systems which are displayed in figure 4-3 (routing provider, air quality provider, traffic provider) trough external platform interfaces. Essentially, the application which is supposed to aggregate data and provide the customer facing service interface has to perform the following steps:

- discover the right platform providers for the purpose of providing traffic- and environmental-related, real-time IoT data.
- engage with the platform providers to understand if and how they can integrate the data sources, i.e. understand their external interfaces
- implement both interfaces according to the specification by platform A and B
- integrate an external map provider (this typically does not constitute an IoT system but rather a web service, although an IoT system could also offer such a service)
- homogenize the acquired data sources in the developed service and deploy it through an external interface

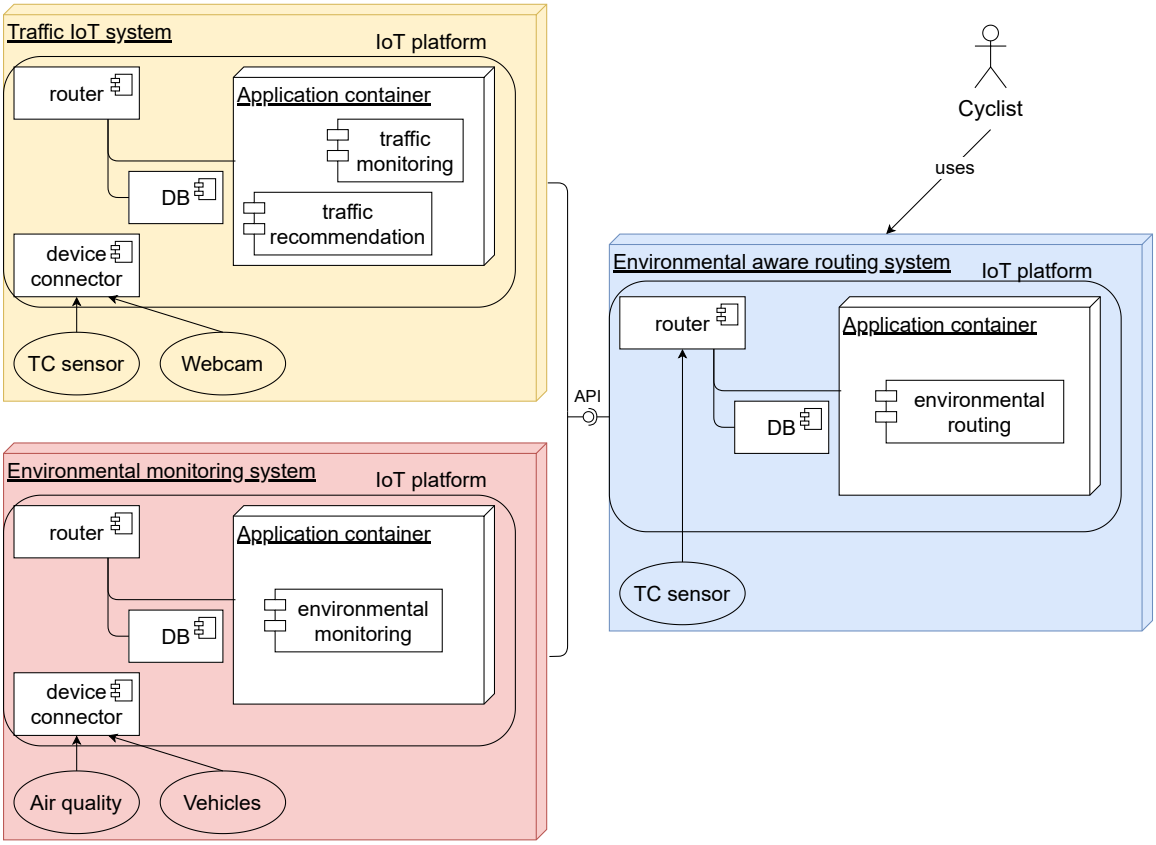


Figure 4-3: A digital service ecosystem for an environmental-aware routing system in a Smart city environment, consisting of a traffic IoT system, an environmental monitoring system and an environmental aware routing system.

Summarizing, these examples exemplify the typical, main interoperability problems in IoT ecosystems :

- The abundance of IoT data & services and the missing link how to find this data & services (especially in open IoT ecosystems)
- Heterogeneous interfaces obstructing data exchange
- Homogenization and alignment of data from multiple sources
- Synchronizing real-time actuation and sensing needs between different systems

4.3.3 Core concepts

To conceptualize an IoT ecosystem, first, the general core concepts of an IoT ecosystem are introduced before going into the details of the dependencies and interconnections between these components. One of the most comprising taxonomies for SoS is given by [CBF+16]. They describe the general components of a SoS in great detail and are repeated where necessary to ease readability. Each of the abstract concepts from the SoS domain will be put into the IoT context as appropriate.

Definition 4.1 (IoT ecosystem) *An **IoT ecosystem** is a System of systems which consists of IoT systems that act as digital species, consuming and providing digital services in an open context environment. It is an arbitrary large community of IoT actors that share the environment and interact, forming a complex, adaptive system of systems. An IoT ecosystem is defined by the network of interactions among its constituent IoT actors.*

Definition 4.2 (IoT system) *An IoT system consists of all components and sub-systems that allow access &management of smart objects and interaction with other IoT systems in an IoT ecosystem through digital services.*

Definition 4.3 (Digital service) *A **digital service** is a service that is delivered through the information infrastructure of a digital service ecosystem. It is provided or consumed by an IoT system.*

In the IoT, a system is a broad concept, since the application domains cover a wide range of areas and also the application and service boundaries are imprecise. We distinguish between **device**, **platform** and **application** level. A typical IoT system usually consists of elements of all these layers, where each layer builds upon the previous. On the lowest level it contains IoT devices, that operate within a physical environment and which are accessed and abstracted trough the platform layer. We can then embedded the idea of "IoT platforms" using the definition of IoT systems. An IoT platform is then a system that connects the application and device level of the IoT reference architecture by offering different functionalities, i.e. access to sensors or data aggregation that are typically offered as services to the application layer. The elements of an IoT platform are thus all software based components that allow it to fulfill its functionalities (e.g. OS, software libraries,...). On the application level, the application and business logic of IoT systems reside. The application level of an IoT system is the abstraction over the platform components. It is what is visible to the outside as a service. The application level provides the digital service to the outside. IoT platforms are usually developed to offer services for a specific purpose, for example a parking spot reservation service, that has a clear scope of functionality.

This service relies on data from smart objects, that would be the parking spots, transmitting data about occupancy. The smart object itself does not fulfill a useful purpose, it is just transmitting data. However, the reservation service is an autonomous system that serves a predefined purpose. An IoT application consists of different elements, which could be a database system, a device management system and a dashboard that are provided by a platform, of which the device management service would abstract the Things to be interconnected. However also other services are used by the application, e.g. the operating system that is offered by the platform to run the application. The software architecture of the application defines the relationships between included components. For an IoT system, it should not be important who provides these components, as long as they offer the needed functionality. Another important property of IoT systems is, that they should not only work together with other IoT systems but also with other systems, for example enterprise systems or specialized software system, which additionally increases the aforementioned issues regarding interoperability requirements.

> **Definition 4.4 (IoT ecosystem environment)** *The entities and their actions in the IoT ecosystem that are not part of an IoT-system but have the capability to interact with the system (based on the system environment definition by [CBF+16]).*

Analogous to the general system's environment definition, and IoT environment contains everything that interacts with the actors that are not part of the IoT system. This can be within but also outside the IoT system's boundary. In general, the IoT environment *"provides an infrastructure that supports the description, compositions, evolution, integration, sharing and distribution of its components, i.e. digital species"* [LBB12]. IoT environments are generally characterized by high heterogeneity and dynamics [RMjP15]. This is a challenging problem for the stability and interoperability of IoT systems, compared to traditional IT systems. Through interoperability, relationships between the components of IoT systems are established, as well as relationships with other IoT systems that allow them to work together and create new functionality that they cannot realize alone.

> **Definition 4.5 (IoT system state)** *"The state of a system at a given instant is the totality of the information from the past that can have an influence on the future behaviour of a system" [CBF+ 16]. In the context of IoT ecosystems, the state of an IoT system usually comprises information about itself (which includes knowledge about hardware and software related components), other IoT systems and about the exchanged messages with other systems.*

Communication related concepts For the interoperability between the constituent systems, a focus must be placed on the communication related aspects of SoS which is why these concepts should be established first.

> **Definition 4.6 (Message)** *"A data structure that is formed for the purpose of the timely exchange of information among computer systems" [CBF+ 16].*

> **Definition 4.7 (Channel)** *"A logical or physical link that transports information among systems at their connected interfaces. It is implemented by a communication system (e.g. a computer network)" [CBF+ 16].*

Definition 4.8 (Interface) *An interface represents the point of interaction of an IoT system with other systems in the environment over time. Interfaces are connected through channels.*

Definition 4.9 (IoT message concept) *A message concept represents a desired state change. All concepts at the highest level change the state of the environment. A concept describes an abstract class of communicative actions.*

Definition 4.10 (Intention) *An intention is a concrete instantiation of a message concept .*

In a system of systems, the communication among the individual systems through the exchange of messages is the core mechanism that realizes the integration of the systems [CBF+16]. In the context of IoT ecosystems, messages are usually related to the exchange of sensor data or actuation of machines . Due to the heterogeneity of IoT devices, the message variety is significant. The usual way of communication between two entities consists of instantiating a particular intention and encoding a desired content into a message which is transported along a specific channel, decoded and understood by the receiver. An intention could be for example a request for an environmental routing service or a parking request. The usual communication model falls short in complex IoT ecosystems due to the mentioned incoherences in communication interfaces and semantics. Interfaces occur in various forms in IoT ecosystems, for example REST-ful or MQTT/websocket interfaces. The main communication channel for the Internet of Things is the Internet. However, also other channels are important, specifically physical channels that connect the device data with the IoT platforms. Not only the semantics affects this communication significantly, but also there is potential misalignment regarding the intentions underlying the transmission. This level is typically addressed in the pragmatic interoperability level (see [TC09]).

Definition 4.11 (Computational action) *"An action that is characterized by the execution of a program by a machine" [CBF+16].*

Definition 4.12 (Communication action) *"An action that is characterized by the execution of a communication protocol by a communication system" [CBF+16].*

Definition 4.13 (Communication protocol) *"The set of rules that govern a communication action" [CBF+16]*

Definition 4.14 (Transaction) *"A transaction is a related sequence of computational actions and communication actions" [CBF+16].*

Actions and Protocols IoT systems possess computational and communication actions. IoT related computational actions are for example, the execution of actuation activities on the underlying physical hardware as well as execution of algorithms inside digital services. IoT system usually execute data requests or actuation requests protocols to receive or provide services to other IoT systems. As already described in the background section, there are a multitude of communication protocols in the IoT space which contribute a great deal to the interoperability problem. As in the context of SoS, sequences of computational and communicative actions form "transactions" [CBF+16]. As an example, we consider a transaction to exchange a digital service. In this exchange the transaction would contain a request for the service, a subsequent offering message and message exchange followed by a payment message. The detailed explanation of the process of a digital service transaction can be found in figure 4-4 which was adapted from the general service transaction lifecycle by [Del13]. To exchange messages across the communication channel, the data coming from or entering one IoT system gets serialized and transmitted through a message protocol to the receiving system.

4.3.4 Roles & interactions inside the IoT ecosystem

IoT systems perform different roles in an IoT ecosystem. Generally, one can differentiate the following types of roles: Digital service consumers (DSC), Digital service providers (DSP), Infrastructure service providers (INSP), digital service developer (DSD) and digital service customer (DSCU). The connection between roles and systems is not always 1-1, for instance a DSP role can be filled by **multiple** IoT provider systems. Also the INSP role can be filled by multiple systems that form

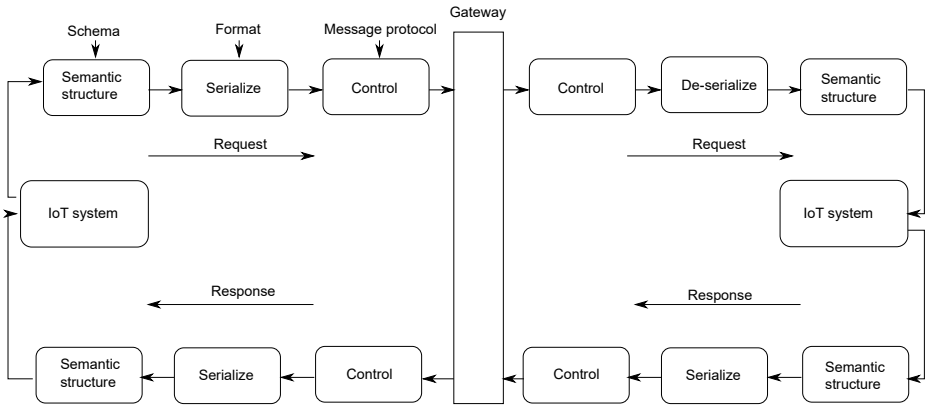


Figure 4-4: The service transaction model between two IoT systems described by [Del13], detailing the process of a service request and response. The client system sends a service request in a specific schema and format through a message protocol to a gateway. The receiver receives the request from the gateway and de-serializes the message content. The response is handled the same way but in the opposite direction.

a conglomeration to build the INSP. Also, an IoT system can operate in multiple roles at the same time!

Digital service consumer - DSC The digital service consumer (DSC) represents any system which requires an IoT related service for operation. It could be anything from a simple client application that crawls IoT data sources to a fully fledged web-based dashboard which combines IoT data sources and can be used to control IoT devices in the physical environment. The digital service consumer needs to find a fitting digital service which can support its needs. More specifically, it needs to perform the following tasks:

- specify its required demand for services and forms queries for service offerings accordingly
- evaluate available service offerings and compare cost & benefit when deciding which digital service to consume
- negotiate the usage terms of digital services with digital service providers using the available infrastructure
- subscribe to and consumes the digital services through the exchange of communicative actions
- potentially unsubscribe from a digital service (in the dissolution phase of the interoperability lifecycle)

Digital service provider - DSP The digital service provider forms the counterpart of the digital service consumer since he offers a digital service to be consumed by other systems. He is also responsible for the proper functioning of the service. In return, he can monetize IoT assets via the service, e.g. providing a data stream to consumer data from a car. In particular, the tasks for the DSP are:

- create a service offering for its digital service that allows DSC to query its service
- negotiates the usage terms of its digital service(s) with DSC using the available infrastructure
- provides the digital service according to the negotiated terms of usage

Infrastructure service provider - INSP The infrastructure service provider is responsible for establishing and operating the IoT environment in which DSC's and DSP's systems operate. Therefore he needs to :

- provide the infrastructure that established the digital ecosystem of services underlying a demand & supply pattern for digital services
- provide a functionality to offer digital services to IoT systems inside the DSE
- provide a matchmaking functionality to connect DSC and DSP
- provide a negotiation functionality for DSC and DSP
- provide a semantic knowledge base for the IoT systems inside the ecosystem
- monitor the interactions between consumers & providers and the (un-) availability of digital services

Digital service developer - DSD A digital service developer (DSD) is the developer of a DSC,DSP or INSP system. He is responsible for implementing the underlying system logic by utilizing the IoT platform services.

Digital service customer - DSCU A digital service customer (or user) (DSCU) is the customer which uses an IoT DSC system to fulfill a certain need.

As an example for a composition of these actors in an IoT ecosystem, consider the earlier introduced use case of the environmental-aware routing service. The IoT ecosystem in this case could be chosen arbitrary large, but here we consider it to consist of all IoT systems in a given Smart city. In this use case, there is one DSC, i.e. the environmental aware routing system which consumes the digital services from two providers, the traffic IoT system and the environmental monitoring system. Consuming in this case refers to exchanging transactions to receive data on air quality and traffic information in the city. The service use respective interfaces in order to transact this information. The infrastructure service in this case could be a digital smart city marketplace platform which provides the infrastructure for the services and offers the mentioned services. A DSCU of the environmental aware routing system in this case could be cyclist who uses the service on his smartphone to stay informed about areas of bad, health-impacting air quality in his city.

Based on this understanding of concepts and roles, it is straightforward to describe the interoperability problem in IoT ecosystems.

> **Definition 4.15 (Interoperability problem in IoT ecosystems)** *An interoperability problem in an IoT ecosystem occurs when a transaction between at least one DSC and DSP system does not lead to the intended IoT ecosystem state change.*

A more detailed formalization of the interoperability problem, aligned with the LCIM metric, will be presented in the next section, but the main theme as outlined here stays the same, i.e. interoperability as a mis-alignment between intended and actual environmental state changes. For example, if the digital service provider in the environmental routing example has no concept of a *"routing request constrained by environmental data"*, it might not infer, based on the state knowledge about the ecosystem and specifically based on the consumers context state, that air quality information should affect the routing response. Hence, a general routing response will not satisfy the consumer of its service. In this case the intended and resulting state of interoperability are misaligned. We can find other examples of such misalignments all over the IoT space which furthermore underlines the importance of interoperability.

Figure 4-5 presents a domain model for IoT ecosystems which connects the previously introduced concepts in a graphical representation.

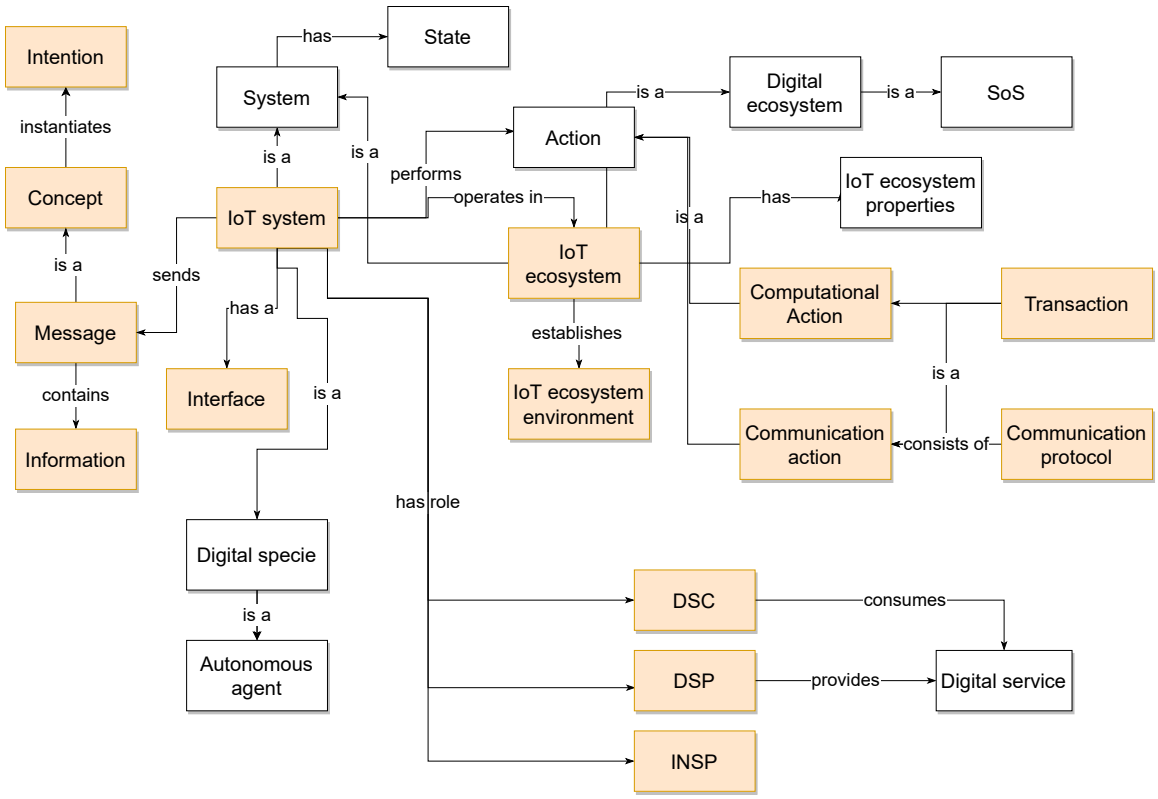


Figure 4-5: A domain model for IoT ecosystems showing the previously introduced concepts in orange including the relationships between the concepts.

Figure 4-6 illustrates the transaction process and transformation of IoT ecosystems in more detail.

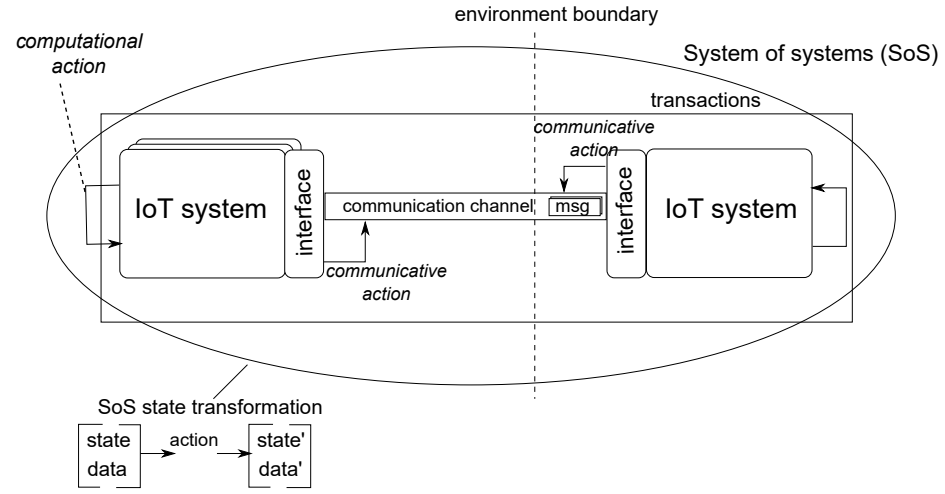


Figure 4-6: Visualization of the dynamics and characteristics of an IoT ecosystem. IoT systems act as service providers (DSP) or service consumers (DSC) and exchange information through a communication channel. An interaction between multiple IoT systems is encapsulated by a transaction which will result in the provisioning and consumption of a digital service. Computational and communicative actions affect the state of the SoS. Each IoT system is separated from from the rest of the SoS through the environment boundary.

4.3.5 IoT Ecosystem properties

Since the IoT ecosystem is described through general SoS and DSE concepts, also the general properties of SoS and DSE apply to IoT ecosystems which are explained in the following.

Self-organization

A self-organizing system consists of multiple components that can change their interrelations. Digital ecosystems are self-organizing systems which can form different architectural models through group intelligence, where local interactions between agents determine the global behavior [LBB12]. The "laws" of an ecosystem, influence the ruling and overall dynamics of the ecosystem and the interaction among individuals of different species [VMZ10]. In the case of an IoT ecosystem, any providing system can be replaced in a collaboration with another service providing similar functionality. These changing interrelations can originate from the providing or consuming system. Also, there is no pre-determined architecture. In the motivating example of this chapter, the environmental-aware IoT system just specifies the goal state to achieve. No system in the example overlooks the whole interoperation, they only know with whom they are directly communicating, so the whole system consists of local interactions. According to systems

theory, a system is autonomous. That means, it is free to fulfill its purpose and free to change its roles at any point in time. Roles are hierarchical and while some roles apply to both provider /consumer and infrastructure provider but some only to one of them.

Loose coupling

Similar to the SoS characteristics of independence between the systems, loose coupling refers to the fact that participating IoT systems in the environment do not depend on each other and the interaction between the species is based on loose contracts and the exchange of communicational actions. The object of negotiation between IoT systems in the IoT ecosystem is a digital contract between a DSC and DSP system. The digital contract represents the specification of the digital service between the two parties and is validated by the provider and consumer regularly to verify the terms, by comparing the data that is sent to the contract terms. This will be referred to later as the *Interoperability contract*.The underlying SoA based architecture of DSE, by definition, allows such loose coupling.

Open

With openess, there is no predetermination which systems will be found inside the ecosystem thus each system must be prepared to interoperate with any other. For example, a DSC can not anticipate which particular digital services and DSPs will be available and which services will enter or leave the environment at runtime. This is because, the IoT-DSE is an open community, and there is no centralized control or fixed roles. Openess, although essential for the success of the IoT, presents one of the main problematic criteria for interoperability.

Domain centered

IoT systems usually perform in their accustomed domain, however they could also operate in neighboring domains if necessary. Based on the generality of an IoT system it can serve multiple purposes into different domains. The interoperability requirements change with each domain application. Some common domains are for example smart city, energy or building technology [IoT].

Demand driven

The digital ecosystem supports the automatic combining of numerous agents (which offer services), by their interaction in evolving populations to meet user requests for applications,in a scalable architecture. Demand for a digital service originates from customer requirements. DSP then

try to create a desired digital service to meet the demand. Without demand, the development of digital services has no meaning, since there would be no system to consume them. What drives the collaboration is the mutual interest among the service providers - there needs to be a willingness to collaborate, to realize a benefit for the customer. Thus, the ecosystem is dynamically created or adapted around a specific demand! The demand-driven property also entails, that species join(or leave) the ecosystem on their own interest. Those systems that work effectively together to fulfill current demand will remain to work together.

Belonging

Each participating IoT system is operated by a different IoT provider for a specific purpose. The purpose can be bound to provide a service in a particular service collaboration or to simply deliver data to whoever may need it. In general, in order to make a system collaborate with other systems inside a SoS, a clear motivation for them has to be realized. [SBV10] refer to a cost/benefit basis. For a system inside the SoS this means: Why belong to the SoS? Who manages the belonging? What happens if the system leaves the SoS, will it break the SoS or the system itself? These questions directly translate to interoperability inside IoT ecosystems : When cross-platform/cross-domain interoperability enables the creation of SoS, how do the IoT systems decide about their collaboration partners? Since in SoS, there is no central coordinator, the systems need to possess the necessary intelligence in order to decide about this themselves. Since they are responsible for their own performance and functionality, they are also responsible for the decision, if they want to collaborate with other systems **at runtime** or not. In SoS this decision is not made by the designer (as in the presented example) rather it needs to be decided autonomously by the participating IoT systems themselves at runtime. In the given motivating example, the purpose of the SoS is to provide an environmental routing service however the difference to a manual integration approach is, that the participating systems engage in a loosely coupled collaboration and remain autonomous in their own right to operate as designed. This is in contrast to the manual integration approach where system only exist for the purpose of the collaboration and they do not fulfill a useful purpose on their own.

Dynamic connectivity

Connectivity, directly translates to achieving interoperability amongst the legacy systems and possibly additions of new systems to the SoS. The systems themselves decide the level of connectivity to other systems (either already present in the SoS or added later). That is a natural consequence

from the fact that IoT systems should be considered autonomous and decide about belonging to a collaboration themselves, as explained before. The fundamental difference in system design for SoS is, that the relationship between the elements (i.e. CS in case of the SoS) is not designed simultaneously with the design of the CS [SBV10]. Also, consumers and providers can become offline or experience outtakes at any time, when connectivity breaks down.

Emergence

Emergence is created through the above mentioned characteristics inside a SoS, i.e. autonomy of the CS, and the dynamic connections that are formed at runtime. It is a characteristic of a SoS, that should not be restricted as it is done in classical system design, since emergent functionality can be beneficial for the SoS and repressing it could lead to inhibiting innovation. The goal in a SoS is thus to create a conducive climate for emergence and quickly detect and remove unwanted behaviors [BS06].

The role of emergence in IoT system collaboration is not well understood. The formation of systems to create an emergent functionality that is not triggered by an external/driving process but rather serves the demand for a specific resource is still a futuristic vision. From an IoT system designer's perspective the questions becomes: *"How will other systems affect my system? Positively or negatively?"*. Therefore, especially with regards to interoperability, it is essential for IoT systems to be actively aware of such emergence and properly react to it.

It is important to understand that these dimensions are not always equally distributed in a particular IoT ecosystem. Rather, these dimensions have to be understood as a scale where certain elements are more or less expressed.

IoT ecosystem properties in context Applying the IoT ecosystem properties to the earlier introduced use cases in the beginning of this chapter yields the following analysis.

Figure 4-7, 4-8 and 4-9 present the unique characteristics of each use case in relation to the IoT ecosystem properties in a graphical fashion. Each dimension in the chart refers to one ecosystem property while the scale ranges from 0 (property not expressed) to 10 (property maximally expressed).

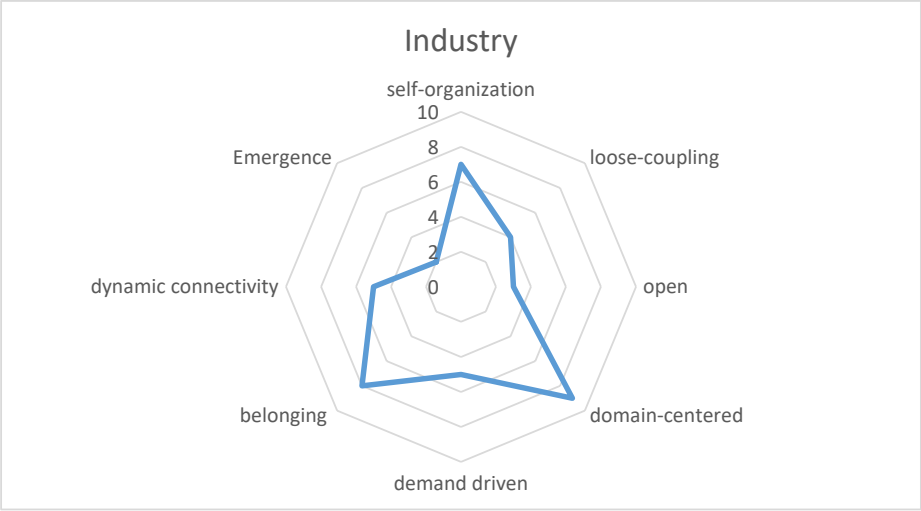


Figure 4-7: Representation of the smart production use case in terms of IoT ecosystem characteristics on a scale of 1 (weak expression) to 10 (strong expression).

Starting with the smart industry use case, it shows high levels of belonging and domain-centricity and medium levels of self-organization. Emergence and openess however are rather under-expressed.

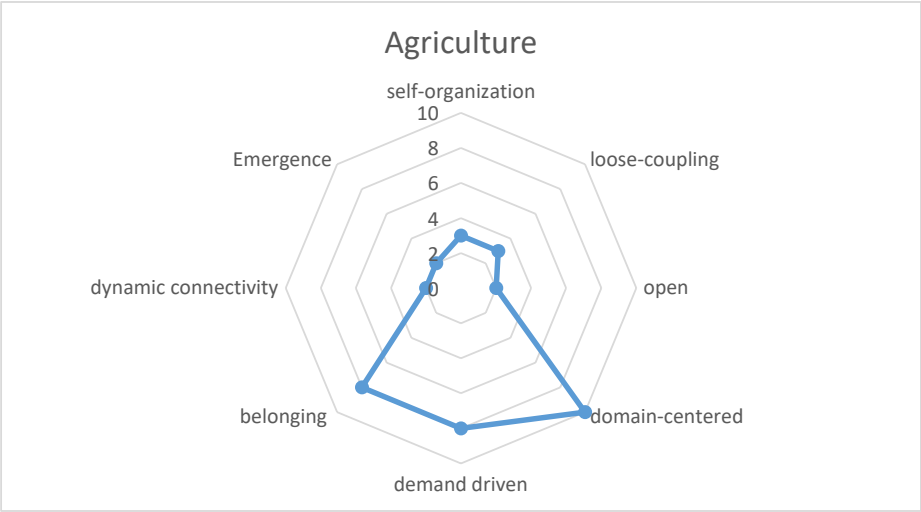


Figure 4-8: Representation of the smart farming use case in terms of IoT ecosystem characteristics on a scale of 1 (weak expression) to 10 (strong expression).

The smart agriculture use case meanwhile illustrates a strong focus on belonging, demand-driven and domain-centricity. Emergence, self-organization and loose coupling are only weakly expressed here. Also dynamic connectivity is not relevant in this case.

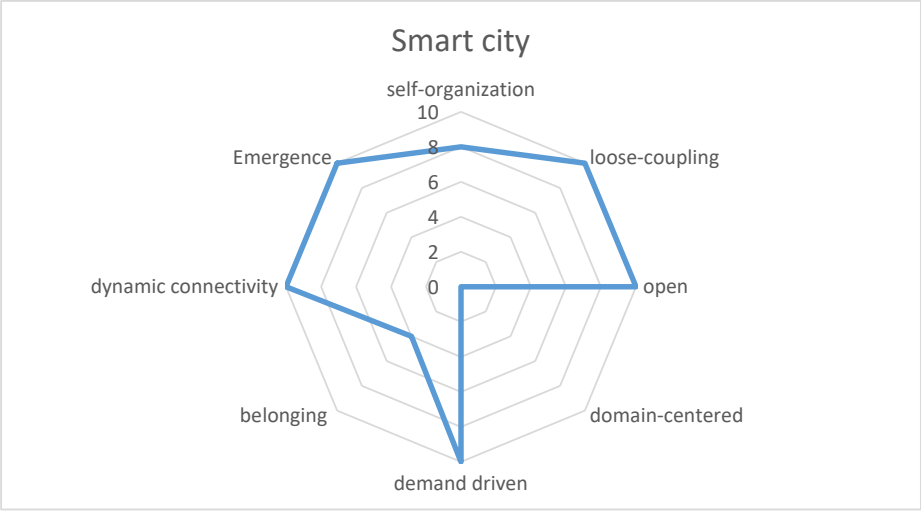


Figure 4-9: Representation of the smart city use case in terms of IoT ecosystem characteristics on a scale of 1 (weak expression) to 10 (strong expression).

Lastly, the smart city environmental routing use cases experiences the complete opposite of characteristics compared to the smart agriculture case. Here, clearly the focus lies in emergence, loose-coupling, dynamic connectivity and openness while belonging and domain-centricity are weakly expressed. This naturally falls into the characteristics of Smart city environments which are highly dynamic and open in nature. All in all, this clearly demonstrates that the characteristics of IoT ecosystems are highly variable and use case dependent which makes a runtime oriented interoperability approach preferably which will thus be the focus of the next chapter.

4.3.6 Centralized vs. de-centralized ecosystems

Traditionally, IoT ecosystems are developed in a centralized fashion which means that there is a central instance which acts as the infrastructure provider. This can be seen from the various examples of IoT platforms and marketplaces which have been developed in the recent past (refer to chapter 2) which act as a central coordinator in the interoperability problem. However, there has been a recent trend to also consider decentralized IoT ecosystems, i.e. without a central coordinator. Regarding interoperability, the decentralization of IoT ecosystems poses a significant problem, since the controlling mechanism for interoperability (e.g. standardization body or central middleware) is not a centralized institution anymore but the process needs to be performed collaboratively in a decentralized setting.

A structured approach to compare centralized and de-centralized IoT ecosystems is to consider

the different roles and tasks in the IoT ecosystem and consider how a centralized vs. decentralized structure affects these roles.

In a **centralized environment**, the INSP is a centralized entity (for example an IoT marketplace provider). This means, that a single platform provider usually offers the infrastructure to implement the IoT environment. All system that want to interoperate need to connect to this central INSP. Rules for exchange are defined by centralized authority (marketplace provider) and can be changed at will. This environment requires less effort to implement and maintain, since it can be controlled through one organizational entity. But, this approach still requires runtime oriented solutions, depending on the requirements maybe to a lesser extent then a decentralized environment.

In a **decentralized environment**, the INSP is de-centralized and the functionality for matching, negotiation and transactions is completely decentralized to the interoperating systems. There is no centralized authority i.e. rules inside the ecosystem are "democratized". They are jointly introduced, agreed upon and enforced by all participants of the network. Implementing this environment is significantly more difficult, since it can not be enforced by a central entity - relies heavily on runtime oriented concepts. With regards to a technical approach for decentralized IoT environments, recent advances in distributed ledger technologies (in particular Blockchain and smart contracts) are a prime example. Smart contracts are deployed in the decentralized network and represent autonomous contracts which define the rules of the environment (e.g. matchmaking) while the blockchain is used for tracking data and messages between the systems.

Comparing both ends of the spectrum of IoT ecosystems, it is important to consider interoperability solutions which are able to also operate in a decentralized fashion. Existing solutions are very much oriented towards the centralized model, e.g. centralized middleware platforms that serve as a mediator between disparate systems. Although these solutions will still exist in the future (especially in closed settings), the limitations of centralized ecosystems will remain and increase in significance, especially in highly decentralized and dynamic areas such as the IoT. To reach an open IoT ecosystem, with all its benefits of value co-creation, a similar perspective as with the design of the Internet itself has to be considered.

Thus, in the next section, an interoperability conceptualization is presented which is able to guarantee interoperability also in semi-decentralized and decentralized settings. It will be shown

that the digital service ecosystem model can be used equally well as a framework for interoperability in both settings.

4.4 Summary

This chapter introduced a conceptualization of IoT ecosystems as a digital service ecosystems. The conceptualization follows the ideas of systems-of-systems and digital service ecosystem research which are particularly well suited to model dynamic systems. What can be observed from this characterization that indeed IoT ecosystems are complex systems which need a proper formal model to be analyzed, which incorporates all of these concepts. This framework is used in the next chapter to formalize the runtime interoperability problem in IoT ecosystems.

Chapter 5

A concept for runtime interoperability in IoT ecosystems

5.1 Aim

The previous chapter introduced a concept to model IoT ecosystems. Based on this knowledge, the goal of this chapter is now to create a concept for runtime interoperability inside the context of IoT ecosystems and to evaluate the benefits over existing interoperability solution concepts, which have been mentioned previously. More specifically, the following questions will be answered:

- How to design interoperability for IoT ecosystems
- How to measure interoperability in IoT ecosystems
- How to autonomously improve interoperability in IoT ecosystems

The chapter starts with a discussion regarding the shift in the viewpoint on interoperability in the IoT. This results in a theoretical model of runtime interoperability which serves as the basis for the implementation in the next chapter. Secondly requirements from the interoperability conceptualization in IoT - DSE are derived and a comparison of interoperability solutions is included. The approach will be the input for the next chapter which provides a concrete conceptual model to implement the derived technical requirements for an IoT interoperability model.

5.2 Runtime interoperability in the context of IoT ecosystems

> *"Achieving interoperability between two resources is much more than sending a set of bytes as a message. The interacting resources need to agree on compatible protocols, formats, meanings and purposes, so that a request message sent by one resource produces the intended effects on the other resource and a suitable response message is returned to the former resource"* [Del13].

Interoperability can be conceptualized from different viewpoints as described in earlier chapters. Since we have seen that IoT ecosystems should be represented from the SoS (respectively the DSE) perspective which naturally represent dynamic, open and self-organizing environments this raises the question, which is the *appropriate* interoperability conceptualization in these ecosystems? The characteristics of IoT ecosystems and the dynamic changes to the underlying digital service composition architecture have a profound effect on the requirements for interoperability. Interoperability is not only a problem about standardization/integration and device control as it has been previously focused in the IoT. Rather, interoperability needs to permeate every aspect of the runtime interaction between IoT systems which requires a new perspective on the design of IoT systems and platforms.

IoT system design is usually approached in a top-down process where an IoT architect designs a system architecture and defines necessary interfaces and standards which need to be implemented for interoperability. Current research on IoT interoperability therefore mostly concerns integration aspects on the level of syntactic or semantic interoperability. In this way, systems can exchange and understand the meaning of information. If these standards and interfaces are not implemented by other systems, no interoperability will be possible. The underlying assumption is that partners will agree on common standards to make systems interoperable. This will only work if both sides of an interoperation follow this top-down approach. But, if we consider that IoT systems need to communicate with other IoT systems in an open context to reach a common goal, then the mere understanding of the **meaning** of exchanged data is not enough [ZTP14]. Furthermore, in the roll-out step, when an IoT system is deployed into production, the systems are currently unaware of their surrounding context and thus unable to recognize and adapt to changes autonomously. This violates the requirement for autonomy and self-organization in IoT ecosystem. The only option for changing the behaviour of a *deployed* and *running* IoT system is

through manual adjustments by the developer, i.e. no runtime data is utilized by the interoperability mechanism itself. This manual adjustment requires large efforts and costs for IoT organizations and potentially (depending on the response time) a reduced user experience. In case of tightly-integrated systems, interoperability problems can easily lead to a complete system failures in a collaborative context which damages the reputation of IoT providers and also risks the adoption of further IoT technology.

These mentioned problems occur, since it is difficult, if sometimes not impossible, for a developer to foresee the interactions of the developed IoT systems with other IoT systems during runtime, due to the open/heterogeneous nature of the IoT ecosystem. IoT systems can be employed in various different circumstances. An essential property of the infrastructure in IoT ecosystem is that they support a certain level of interoperability between the ecosystem members. This is required so that IoT systems are actually able to detect other IoT systems inside the ecosystem, which is a pre-requisite for interoperation. Increased complexity of system design and resulting emerging instability makes existing interoperability principles not fully sufficient to provide the needed overall awareness, integrity, and pursuit of global goals, with runtime adjustment to new ones. The situations are often complicated by the necessity to operate in spaces with high connectivity and inter-dependence, also with numerous actors having own, often quite different purposes and interconnections. Essentially the system only works as intended if the situation that are experienced at runtime align with what has been designed by the developer before [TPB+11]. With regards to interoperability this means, interoperability will only be established with those systems whose interfaces and operation modes are understood and aligned with the developers initial vision. If the system composition changes, there might be a degradation or complete lack of interoperability. Also, goals of the customer of the IoT system might change over time which might require reorientation of the system's collaboration partners. These are only two issues that highlight the need to put the focus of IoT-interoperability research on the runtime aspect of the interoperation, i.e. all those aspects that may occur after the initial deployment of the system. An evolution of interoperability concepts is missing, from a purely design-time problem to a runtime problem! Essentially this addresses the characteristic in digital service ecosystem of moving from tightly to loosely coupled systems and from closed to open systems [TPB+11].

The focus in digital service ecosystem (per definition in IoT ecosystems) needs to be *"on **dynamic**, **behavioural** and **conceptual** interoperability and interactions between services, and between humans and services"* [IOKP16]. Interaction between services refers to the direct interactions between digital services whereas the interactions between humans and services refers to the fact, that digital services can be also directly exposed to the end user which is then used in the specific end user context. Dynamic behavior is then the prerequisite for increased user experience and the creation of value added services, on the other hand dynamic behaviour is essential in order to react to situations not anticipated at design-time of a system, thus increasing resilience. Attributes such as *"awareness"*, *"intelligence"* and *"extroversion"* [ZTP14] enable the IoT systems to behave and communicate autonomously. But, if the systems are built from heterogeneous systems, that are autonomous and not controllable by one entity, this raises the point of who controls the interoperability between IoT systems? If a central entity is responsible for interoperability this will violate the autonomy of system operators since they need to comply with whatever this central instance requests. On the other hand, if there is no such centralized control, IoT systems must possess the necessary capabilities to address interoperability problems without external help. This latter view is more in line with the underlined nature of the IoT as an inherently unbounded and decentralized system. IoT systems are very diverse and can be used in different context and use cases where varying requirements are imposed on them. For example, a smart camera system can be used for people tracking purposes as well as for traffic analysis using computer vision algorithms. Also, IoT systems can be upgraded during operation to possess new or different functionalities that cannot be anticipated by other, collaborating systems, requiring them to adapt to new configurations. IoT systems are generally expected to work together with other IoT systems to provide value added services, making them unbounded in nature [CFMP05] and hence requiring them to be able to distinguish good from bad collaboration partners. [AO16] stress the fact, that the quality of a digital services is very difficult to achieve with increased ecosystem size as the supporting services need to provide the needed quality as well. Interoperability is currently **not** or in-completely handled by the majority of methods to create digital service ecosystems - especially considering dynamic and pragmatic interoperability [AO16].

Regarding the definition of interoperability in the first part of this chapter, it can be substantiated based on the previous argumentation.

Definition 5.1 (Runtime interoperability in digital service ecosystems) *Runtime interoperability is the ability of IoT systems to establish and maintain interoperation with other (potentially unknown) IoT systems requesting (providing) digital services autonomously according to underlying intentions and goals. Runtime interoperability is achieved when IoT systems are able to jointly fulfill a common mission (or goal), for a non negligible amount of time through the interoperability lifecycle.*

Since the IoT ecosystem concept is conceptually rooted in the SoS domain, it is obvious to draw analogies for the conceptualizations of interoperability from SoS literature as well. Most studies on interoperability in SoS restrict the characteristics of SoS to their respective domains, with little cross-domain definitions. One of these cross-domain studies by [SBV10] defined *"evolutionary development"*, *"emergent behavior"*, *"self-organization"*, *"adaptation"*, *"complex systems"*, *"individual specialization"*, and *"synergy"* as important characteristics of SoS. They characterize interoperability in SoS as *"Making disparate, diverse, autonomous, and synchronized entities work together without losing their individual sense of purpose and without loss of idiosyncratic capability, in order to realize some higher level otherwise unattainable purpose"* [SBV10]. This already addresses the key point for interoperability in Systems of systems - the individual systems need to be considered as autonomous, rational agents that are acting on their own benefit but at the same time realize a need to work together with other systems to achieve their goals. Interoperation in the case of SoS thus means, that the individual systems *"exchange flows (data and services, material or energy)"* while *"preserving their autonomy"*[BD15]. A particular challenge for the formation of a SoS is the high heterogeneity of their constituent systems, which are distributed, independent, and developed with different technologies and for diverse platforms - essentially a complex, distributed system. Therefore, abstracting away the inherent heterogeneity of these systems and making them interoperable are imperative issues to be tackled in SoS architectures. Interoperability in SoS needs to be placed around high-level descriptions of goals and of services since generally, the designer has to work with black-box systems, where the inner workings of the collaborating systems are not known [Luk15]. According to [Fis06], successful interoperation in SoS can not depend anymore (as in traditional integration activities) on a centralized institution which issues *"centralized control dependent on global visibility and coordination among predictable error-free components in predetermined situations"*. In SoS, autonomous constituents need to *collaborate* to fulfill an over-

all system purpose, i.e. the **governance** of interoperability is decentralized. The effectiveness of this collaboration relies on two key factors: *"the degree to which the interaction partners share a common purpose"*[Fis06] and *"the autonomous freedom each parter has to fulfill this purpose"* [Fis06].

[MDC10] define the following requirements for interoperability which can be used as a rough framework to capture interoperability in the IoT ecosystem.

- Compatibility *"A compatibility requirement is defined as 'a statement that specifies a function, ability or a characteristic, independent of time and related to interoperability barriers(in Enterprise context: (conceptual, technological, and organizational)) for each interoperability concerns (data, services, processes and business), that entities must satisfy before collaboration effectiveness'"*[MDC10]
- Interoperation *"An interoperation requirement is defined as 'a statement that specifies a function, ability or a characteristic, **dependent of time** and related to the performance of the interaction, that entities must satisfy during the collaboration'."*[MDC10]
- Reversibility *"A reversibility requirement is defined as 'a statement that specify functions, abilities or characteristics related to the capacity of entities to retrieve their autonomy and to return to their original state (in terms of their own performance) that they must satisfy after collaboration'."*[MDC10]

Comparing this requirements framework to the usual approach to interoperability in IoT, we observe that usually only the **compatibility** level is addressed, through standardization. This level only deals with the design time aspect of interoperability since it is time-oblivious. The interoperation and reversibility level on the other hand relate to the runtime aspect of interoperability. It was already explained that interoperability in complex systems such as the IoT cannot be reached through compatibility level approaches alone due to the nature of the IoT which in its very sense relates to the operational context in which systems find themselves. For example, since IoT environments are dynamic and pervasive, updating and managing semantic descriptions becomes and important issue [VF13]. Also, the IoT does not only involve exchange of information but also **actuation** in the physical world. This means, that communication errors due to missing or falsely interoperability will have direct consequences making reversibility an important issue.

The importance of a runtime approach to interoperability calls for runtime-oriented concepts in the design for IoT systems. Especially, the following questions need to be answered:

- How do IoT systems detect changes in their environment?
- How do IoT systems decide with whom to collaborate if this decision is made at runtime?
- How do IoT systems react when finding potential partners for interaction - how do they engage?
- How do IoT systems decide when and how to adapt (or disassemble) the state of interoperability with other IoT systems?
- How do IoT systems keep track of adaptation changes in order to determine if and how to reverse them to their original state?

This list is by no means meant to be exhaustive but it motivates that in order to reach these objectives, new **interoperability** capabilities and mechanisms for IoT systems and platforms are necessary to create these envisioned dynamic systems that are able to interact with their environment seamlessly without human advisors and react automatically to perceived changes. When considering the scale of the IoT, not only among the billions of devices that should be connected but also the number of IoT platforms, the requirements proposed by [Gué14] become more and more valid since frequent changes between interoperating systems will become more frequent, rather than an exception. Also, failures of systems have to be considered. Thus a complete interoperability lifecycle approach, as motivated in [CD12], will become necessary for interoperability in the IoT! In the course of this work a concrete approach to solve this problem is presented. But first of all, a more conceptual approach to the runtime interoperability problem is required.

5.3 Requirements on IoT system design for runtime interoperability

Based on the previous concept for runtime interoperability we can now answer question about the requirements to establish interoperability since we have a complete system model and can identify those elements and interfaces that are essential to maintain interoperability in this model. The primary difficulty lies in a way to define measures for these requirements. Each requirement will thus also contain an *acceptance criteria* that is measurable in the later introduced simulation. In the latter evaluation stage these requirements should evaluated as to whether they are met by the proposed IoT system architecture .

Service orientation and discoverability Service composition and service discovery are functionalities which are essential in IoT ecosystems, where IoT systems offer digital services to other IoT systems [RMjP15]. Service orientation is addressed by a service-based implementation of the interoperability lifecycle where the systems way of communication is through the exchange of services. A centralized and semi-decentralized lifecycle implementation provide an easier service discovery, from the perspective of a system designer, due to the the centralized point for querying and registering services. However, existing and discontinued approaches from the webservice domain such as UDDI [Raj11] have already proven that a central store for all services is difficult to maintain due to scalability reasons. A decentralized implementation would be advantageous to its better scalability behaviour however it is significantly more difficult to implement, since systems need to synchronize state of all available services in the system.

Discoverability plays a profound role in open IoT ecosystem with evolving populations and is tightly integrated with service orientation. IoT systems are expected to work together with other IoT systems in an ad-hoc fashion, i.e. without prior knowledge about the interaction on both sides. Hence they need to possess mechanisms to search other interoperation partners at runtime, filter partners and decide to engage potential partners and negotiate on a model of cooperation. The search process needs to return interoperability characteristics which are analyzable in an automatic fashion by the IoT systems. Thus, a service layer ensures that IoT systems can find an exchange services (and thus data and functionality) amongst another.

Definition 5.2 (Service-orientation requirement) *IoT systems need to be able to expose or consume functionality through a service abstraction layer.* ***Acceptance criteria****: An IoT system is considered service-oriented if all of the interoperability lifecycle functionalities are accessed through a service interface and all its offered services are offered through a service interface.*

Definition 5.3 (Discoverability requirement) *IoT systems need to be able to find interoperation partners autonomously at runtime, without prior manual interface configuration. Discoverability can be differentiated between active and passive.* ***Acceptance criteria****: An IoT system acting as a DSP fulfills the passive discoverability requirement when it is discoverable for any other IoT system in the environment. An IoT system acting as a DSC fulfills the active discoverability requirement when it is able to discover any other IoT system in the environment.*

Semantic interoperability mechanisms IoT data and services are usually described in various different ways which results in semantic mismatches during communication. To provide runtime interoperability, information needs to be processed semantically with the help of ontologies. For this processing, a centralized lifecycle implementation is advantageous since semantic information (ontologies and semantic translations) can be stored in a centralized location. In a decentralized implementation, each IoT system needs to be able to understand and translate ontologies from all other IoT systems. However, a centralized implementation also means, that all data is routed through a central infrastructure which might be impractical for real-time and privacy sensitive applications. For this reasons, a semi-decentralized implementation joins the benefits of keeping the data exchange between the IoT systems while providing a central place for storing semantic related information.

Revisiting the semantic based requirements by [CFMP05], one can identify a plethora of solutions regarding **semantic interoperability** in IoT. It is a common point of agreement that semantic technologies provide the necessary **foundation** for interoperability on the semantic communication level ([SBC+15]).

Definition 5.4 (Semantic communication requirement) *To handle information semantically, IoT systems need to be able to process messages from other IoT systems using ontologies and use ontologies to map outgoing messages semantically.*
***Acceptance criteria**: An IoT system incorporates a semantic communication mechanism when for a message m and an ontology O the semantic communication mechanism $p(m, O)$ leads to the desired internal state change by the sending system.*

Pragmatic interoperability mechanisms In order to reach pragmatic interoperability, information flows between IoT systems need to be interpreted context-based and interpreted based on the purpose of information usage instead of just based on the objective content. To determine the usage context, the systems need to have knowledge about the current context of the other IoT systems and the environment in their interoperation. This entails, for two different contexts, the interpretation of semantically identical messages may be different or identical.

Definition 5.5 (Pragmatic communication requirement) *To handle information pragmatically, IoT systems need to have mechanisms in order to process information with respect to a concrete usage context. **Acceptance criteria**: An IoT system incorporates a pragmatic communication mechanism when it incorporates context information c into the semantic communication mechanism p so that for a message m the reaction $p(m, O, c)$ leads to the desired state change.*

Decisional & operational autonomy From an internal systems perspective, autonomy can be differentiated into *"decisional autonomy (the system assures its governance and remains always able to decide actions) and operational autonomy (the system remains able to preserve its performance in terms of costs, delays and quality of services)"* [CD12]. The IoT ecosystem is built up of heterogeneous systems, that are **autonomous** and not controllable by one entity [CFMP05]. This raises the question of who controls the interoperability inside the ecosystem? If there is no central control, IoT systems must possess the necessary capabilities to address interoperability problems dynamically at runtime.

The definition of autonomy already hints to the fact that a centralized lifecycle implementation is not the optimal solution since a centralized implementation usually restricts systems

decision freedom, limiting their ability to act on their own behalf also to unforeseen situations autonomously. Autonomy is increased by moving the responsibilities inside the lifecycle functionality away from a central instance towards the participating systems. For example, in the semi-decentralized architecture, autonomous system behaviour is facilitated since the transaction functionality is distributed to the individual systems. The best solution for maximal autonomy would see the complete interoperability functionality being decentralized to the participating systems without a centralized INSP system. Although this has the advantage of maximal autonomy, it brings its drawbacks with respect to performance, coordination and security since there is no instance controlling settlement or service discovery for example.

IoT systems are very diverse and can be used in different context and use cases where varying requirements are imposed on them. For example, a smart camera system can be used for different purposes, such as people counting or tracking or traffic analysis. Also, IoT systems are usually being upgraded during operation to possess new or different functionalities that cannot be anticipated by other, collaborating systems hence they will not be aware of that fact until they get in contact with the upgraded system. They are generally expected to work together with other IoT systems to provide value added services, making them unbounded in nature [CFMP05] and hence requiring them to be able to distinguish better from worse collaboration partners.

Autonomous decision making is important for systems to self-protect when engaged in interactions with other systems at runtime, which are unknown at design-time. Concretely, in technical terms, it would mean for a system to be able to maintain its response times even if another system is trying to flood it with requests. The question is, if there is no *controlling entity*, on which basis can IoT systems make such decisions autonomously?
There has to be a controlling mechanism that makes sure the system remains functional even if it changes interactions partners quite frequently during the evolution of the interoperation (operational autonomy). Interoperability and a shared understanding between components about their behaviour is necessary to allow ad hoc applications composed by end users. Lacking interoperability can result in mission critical failures, especially in Industrial IoT context! For example, in the Industrial IoT, Cyber-physical systems are used which increase the variance in the behaviour space of the overall system. Since responses to external stimuli are state dependent, the behaviour is not deterministic. Because Industrial Internet systems are large-scale distributed systems of multi-vendor machines, safe, secure and resilient operation is required. The systems must be able

to make individual decisions based on the information available, requiring a local decision process in each system [XLZ04].

> **Definition 5.6 (Autonomy requirement)** *Interoperation can have drastically different time spans and no central interoperation operator can be expected to deal with interoperability issues at runtime. Thus IoT systems need be able to react to interoperability situations autonomously.* ***Acceptance criteria****: A solution meets the autonomy requirement if for a given system state change function $f(s, a) = s'$ where $s \in S$ for all possible states, and a given IoT system's utility function $u(s)$ the system is able to perform action $a \in A$ so that the following equation holds : $u(s') >= u(s)$*

Interoperability reasoning & measurability Relationships between systems play an essential role in this thesis since effectively, relationships are closely linked to interoperability. When *systems* or *system elements* are interoperable, they form relationships among each other and exchange information. This accomplishes the desired functionality of the overall system and emergent behaviour inside the system [NGC09]. In the IoT, especially the dynamic creation of relationships during the lifetime of IoT systems is essential. This can only be achieved, when IoT systems transform from merely providing functionality in a collaboration to be actively aware of the collaboration or interoperation by possessing the necessary competences to reason about interoperability related decisions at runtime and for example deciding when to participate in a particular service collaboration. To be able to reason about interoperability, it needs to be measurable. To measure interoperability (and also efficiency from the previous requirement), **metrics** are necessary to analyze semantic and pragmatic interoperability runtime. The metric needs to be based on the notion of interoperability utility which is always created through use of a service in a particular context. Quantitative metrics for interoperability have been developed in the past, e.g. by [FCGD07] using the *iScore* which can be used to quantify the amount of interoperability between a number of systems. The question remains, in the context of runtime interoperability, what measurements to apply. From a requirements perspective, the measurement needs to be able to quantitatively analyze interoperability and be usable by IoT systems in an automatic way in order to have some measure of judgment to improve interoperability. The notion of a context based assessment of interoperability makes more sense than a generic measurement since the same systems can have different levels of interoperability, depending on the context they see themselves in. The only

constant in this case is the value which is generated through the interoperability!

Definition 5.7 (Interoperability reasoning requirement) *In order to be able to reason about interoperability related decisions, state information about the current state of interoperability needs to be gathered automatically and be processed in an automatic fashion.* ***Acceptance criteria****: A solution meets the interoperability reasoning criteria if for any given system state s an IoT system possesses a mechanism m(s) to transform the state information into an internal representation and measure the current level of interoperability with other IoT systems.*

Definition 5.8 (Measurability requirement) *In order to measure interoperability, IoT systems need to process internal metrics to quantify the state of interoperability with respect to the communication with other IoT systems, their own state as well as about the state of the other interoperation partner(s).* ***Acceptance criteria****: An IoT system is able to measure interoperability if for any interoperability state* ***s*** *the system's interoperability measurement function O produces an output O(s) which allows to quantify the current level of semantic and pragmatic interoperability with other communication partners. Each state* ***s*** *contains references to other IoT systems so that the subject IoT system can assess the level of interoperability with all other system it interacts with.*

Adaptivity Complex systems are considered adaptive when they are able to change and learn from their experiences [UM09]. [Ack71] describes formally what it means for systems to be *adaptive*. He distinguishes between different levels of adaptation that affect different parts of the system. Generally, a system is adaptive, if a change in the environment and / or its internal state triggers a change in its own state and or that of its environment so as to increase its efficiency with respect to the goal/ goals it pursues. Adaptivity is thus an ability of a system to modify itself in order to keep its efficiency.

[NGC09] have identified the need for systems to adapt as an important characteristic in order to improve interoperability. They state that a system, that is able to self-react to changes in its environment and adapt its internal structure or behaviour accordingly, while keeping its original objective, have a greater interoperability potential.

A relating concept is *reversibility* of systems, which means that any adaptation that was performed in order to enhance interoperability can be reversed. For example, if a system has adapted one of

its interfaces in order to interoperate with another system by adding an additional endpoint or supporting a new protocol, this change can be reversed to the original version, when the interoperation is over. However it would be ideal, if the system could determine, if the changes could be useful in general also for interoperability with other system so that it might in fact keep the changes in tact. This leads to the next desirable property of *learning*.
[Muf09] also mention that service discovery and composition, inherent to the proposed middleware based approaches for IoT, need to be autonomous dynamically **adaptable**.

If we compare this outcome with the interoperability requirements proposed by [Gué14], there are still solutions missing in order to properly deal with the open nature of systems of systems of which IoT is a representative. If a system must change due to changes in services the system uses, an adaptive middleware solution that is build on predefined policies and rules will only work, as long as the potential changes can be anticipated at design-time. Adaptation is the process whereby a system becomes better suited to its environment given its purpose , the concept of adaptation is closely linked to the goal(s) of the system.

Adaptivity first of all requires the system to be modifiable at all. The interoperability lifecycle implementation provides the points of adaptation, e.g. with respect to matchmaking, negotiation or transaction. Generally, with increased decentralization, the responsibility of providing these adaptive actions is shifted towards the individual IoT system. This is advantageous, since usually the system provider has the best knowledge about his system. In comparison, in a centralized implementation all points of adaptation need to be implemented by one provider. One can imagine that this setting restricts the breadth of adaptivity compared to a completely decentralized setting in which each consumer or provider system is offering its own adaptation logic. On the flipside, decentralized adaptation also brings the disadvantage of uncontrolled adaptation if coordination between the systems about their adaptations is not provided, *"The SOI (system-of-interest) operates seamlessly with the other systems of its environment in order to fulfill its mission. It is able to control, adapt or anticipate problems promptly i.e. to reduce impacts on other systems with more or less desired effects and adverse.*"[CD12]

Definition 5.9 (Adaptivity Requirement) *IoT systems need to be **adaptable** to ensure interoperation at runtime. System interactions that are built on predefined policies and rules will only work, as long as all the interactions can be anticipated at design-time. In order to address runtime interoperability, the IoT systems must be able to adapt to system changes that were not anticipated (for example systems that are added, changed or modified during operation).*
***Acceptance criteria**: A solution is interoperability-adaptive, if for any given system state change S a system is able to respond with an adaptive action $a \in A$ that has an effect on its own state and/or the IoT environment state with respect to the goals it pursues.*

Learning Machine learning has a long history in computer science and in recent years, the trend of artificial intelligence has led to increasingly sophisticated learning systems. In systems theory, [Ack71] describe *learning* generally as the increase of one's efficiency in the pursuit of a goal under unchanging conditions. However a prerequisite for learning is that the system has a choice among alternative courses of action. As already hinted, adaptation and learning can also be combined, if the system repeatedly learns to operate more efficiently in a given environment [Ack71].
Especially due to the dynamics of IoT ecosystems, learning to interact in these environments plays a profound role. The learning behavior for IoT systems needs to be based on feedback from the environment which plays an integral part of the learning process. We deal with ad-hoc learning problems, instead of supervised or unsupervised learning problems since the learning context often changes depending on the composition of the IoT ecosystem. This requires a learning process which is able to apply real-time feedback. Feedback about the state of interoperability is gathered and re-training is performed on this data for improved performance. By incorporating this environment feedback, the learning module can become quite robust.

The learning mechanism of the system thus needs to have the following properties:

- Learning needs to be based on feedback from the environment and include new knowledge in the learning process.
- Learning needs to be mostly online since little to no prior training information will be available due to the ad-hoc nature of the IoT environment.

Definition 5.10 (Learning requirement) *IoT systems need to be able to learn to interact/react in/to new situations at runtime, according to given constraints and their capabilities based on feedback from the environment.* ***Acceptance criteria****: An IoT system is considered to learn a behaviour in an environment when its efficiency between timesteps t_1 and t_2 increases by a measurable amount δe through feedback from the environment and actions he performs in the environment.*

5.4 A model for runtime interoperability

In this section, a concept for runtime interoperability is presented based on the definition of runtime interoperability in definition 5.1 and the previously defined requirements for IoT system design to enable runtime interoperability. The model is split into two parts: a model for the interoperability lifecycle and a model for autonomous adaptation of interoperability at runtime.

Since interoperability is essentially a problem of collaboration between multiple IoT systems, the following model for runtime interoperability follows a multi-agent based approach between a list of service consumers (DSC) $\mathbb{C}$ and service providers (DSP) $\mathbb{P}$. In this theory, an IoT system is composed of an I-IOP (**I**oT - **I**nter**op**erability) agent and a system logic part. The latter part contains the IoT platform services that are used for the IoT systems logic implementation. The purpose of the I-IOP agent is to establish interoperability with other IoT systems in the IoT environment. It consists of two components: an **interoperability module** (IOP module) and an **autonomic control module** (AC module)(figure 5-1).

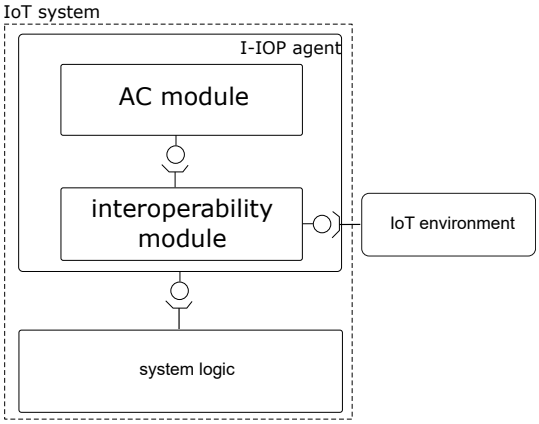


Figure 5-1: Visualization of an IoT system instance, consisting of the underlying IoT system logic, the I-IOP agent (consisting of the AC module and the interoperability module) and the interface to the IoT environment.

5.4.1 The AC module

The AC module of the I-IOP agent model is a generic autonomous control system and will be formalized as follows:

Definition 5.11 (Formalization of the AC module) *The AC module is a tuple* $m =< \mathbb{I}, \mathbb{A}, \mathbb{O}, \mathbb{Z}, \pi, \tau >$ *where:*

- $\mathbb{I}$ *is the set of interoperability information states of the I-IOP agent*
- $\mathbb{M}$ *is an interface to the interoperability module*
- $\mathbb{A}, \mathbb{O}$ *The action set taken by / observations provided to the agent by the environment*
- $\mathbb{Z}$ *is the set of communicated observations* $\mathfrak{z}$ *available to the agent through the communication with other IoT agents*
- π *is the action selection policy:* $\pi : \mathbb{I} \longrightarrow \triangle(\mathbb{A})$
- τ *is the information state update function* $\tau : \mathbb{I} \times \mathbb{A} \times \mathbb{O} \times \mathbb{Z} \longrightarrow \triangle(\mathbb{I})$

The AC module builds up an internal representation of the interoperability information state $\mathbb{I}$ of the I-IOP agent based on the information it receives from the environment $\mathbb{O}$ through the interoperability module with regards to the interoperability process and by interpreting these observations and bringing together the auxiliary and primary state information it received through communication with other I-IOP agents $\mathbb{Z}$. How this information state update process is performed is defined in the information state update function τ and is unique to each agent. The agent is then able to base his decisions on the correct state of interoperability $\mathbb{I}$ with the other agent(s) which consists of the complete semantic and pragmatic information available to the AC module. It uses this information to perform actions on the interface to the interoperability module $\mathbb{M}$ and or by communicating with other agents which in turn affects the interoperability information states. In the optimal case, the actions taken by the agent would be equivalent to one central instance that controls the actions of all the I-IOP agents involved through a joint policy based on perfect knowledge to maximize interoperability. This joint policy can be split into individual policies for all the agents $\mathbb{G}$ involved $\pi_i, i \in \mathbb{G}$. If, auxiliary information is misinterpreted (e.g. through semantic mismatches) the agent receives incomplete or false state information. This inevitably

leads to problems in the interoperability and potentially in a collapse of interoperability. To avoid such issues, a couple of agent communication standards have emerged in the early 2000s which are well suited for this context since they already define semantics and pragmatics within agent communication, for example for negotiations which eases semantic interoperability. A prominent example is FIPA-ACL which was standardized by the W3C [1].

5.4.2 The interoperability module

Interoperability in IoT ecosystems is modeled as a four-step process, referred to as the *interoperability lifecycle*, similar to the service collaboration lifecycle by [RK09] and is shown in figure 5-2. The interoperability module of an I-IOP agent is responsible to interact with the IoT environment through an implementation of the interoperability lifecycle (definition 5.12). It interfaces with the underlying system logic and the surrounding IoT environment through an interoperability lifecycle adapter. The autonomous control (AC) module acts as a meta-system, observing and controlling the interoperability with other IoT systems through the assessment of the current interoperability state. The concrete architecture and implementation of the I-IOP agent model will be explained in chapter 6. The underlying system logic can implement any kind of IoT related functionality, either on the platform or device level. This means, the I-IOP agent can be deployed on IoT platform services and on IoT devices directly.

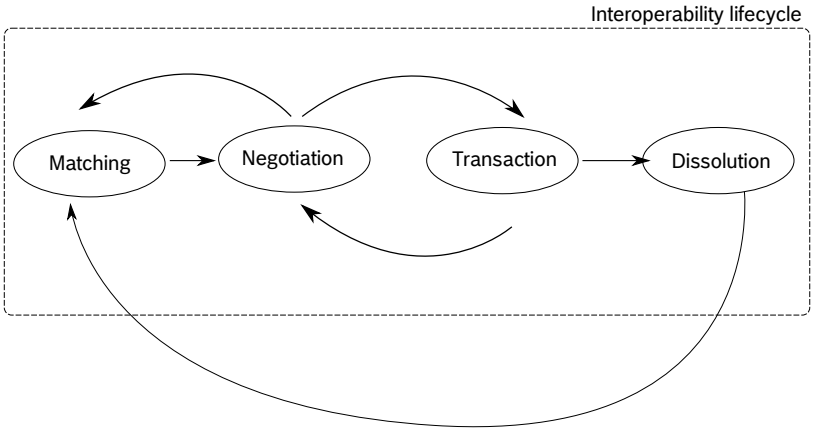


Figure 5-2: Visualization of the interoperability lifecycle, consisting of the phases: matching, negotiation, transaction and dissolution.

[1] http://www.fipa.org/repository/aclspecs.html

Definition 5.12 (Interoperability lifecycle functionality) *The interoperability lifecycle functionality in an IoT ecosystem is a function $\mathcal{F} = (\mathcal{F}_{\mathcal{M}}, \mathcal{F}_{\mathcal{N}}, \mathcal{F}_{\mathcal{T}})$ composed of sub-functionalities $\mathcal{F}_{\mathcal{M}}$ (matchmaking functionality), $\mathcal{F}_{\mathcal{N}}$ (negotiation functionality) and $\mathcal{F}_{\mathcal{T}}$ (transaction functionality) .*

Definition 5.13 (Matchmaking functionality) *The matchmaking functionality $\mathcal{F}_{\mathcal{M}}(\mathfrak{d}, \omega)$ receives as inputs the specification of a service demand $\mathfrak{d}$ and offers for service delivery by potential providers ω. Its output is a list of $n >= 1$ service provider offerings $\{\mathbb{P}\}$ ordered by the degree of matching with the demand.*

The matchmaking functionality is responsible for bringing together different parties of the IoT ecosystem. It can be implemented by a centralized IoT system that has knowledge about all participants of the ecosystem or in a decentralized way, where all ecosystem systems jointly execute the functionality.

Definition 5.14 (Negotiation functionality) *The negotiation process is a functionality $\mathcal{F}_{\mathcal{N}}(\mathbb{C}, \{\mathbb{P}\})$ which receives as input a list of $n >= 1$ provider systems matched with a consumer system $\mathbb{C}$ and negotiates a list of interoperability contracts $\mathfrak{I}$ for each provider system $p \in \mathbb{P}$.*

The negotiation functionality is implemented as a negotiation mechanism, e.g. a specific negotiation protocol such as an auction or a negotiation game ([MVEP03]). The interoperability contract $\mathfrak{I}$ specifies the constraints for the interoperation between the IoT systems. For example it contains details about maximum message delays, price for a digital service or the required minimum utility for the customer. Thus, an interoperability contract represents a concrete instantiation of a negotiation objective function and can be considered a blueprint for an interoperation between DSC and DSP systems. In case the negotiation between the DSC and DSP system fails, the interoperability contract is void.

Definition 5.15 (Interoperability contract) *An interoperability contract is an object defining the following elements for a transaction between two IoT systems:*

- *Identifications of the DSC and DSP IoT systems*
- *A protocol P*
- *The service delivery type t*
- *The underlying ontologies used by the IoT systems O*
- *A price being paid by the DSC to the DSP systems for consuming/providing the service p*
- *Quality of service parameters, such as accuracy and max. delay QoS*

Definition 5.16 (Transaction functionality) *The transaction functionality $\mathcal{F}_{\mathcal{T}}(\mathfrak{I}, \mathbb{S}_C, \mathbb{S}_P)$ delineates the transactions between the IoT systems, i.e. the delivery of the negotiated digital service according to the interoperability contract terms. It offers parameters for adaptation so that the systems comply with the negotiated contract.*

Definition 5.17 (Dissolution functionality) *The dissolution functionality $\mathcal{F}_D$ dissolves an interoperability contract*

The transaction functionality is the most essential part of the interoperation between IoT systems. It deals with the semantics and pragmatics of communication and describes the actual exchange of the digital service. The previously negotiated contract acts as a kind of blueprint for this phase. Note that authentication and payment are part of this phase, however not in scope of this thesis. The next part will introduce an optimization problem in order to optimally configure a transaction functionality towards such a contract.

Definition 5.18 (Interoperability lifecycle implementation) *An interoperability lifecycle implementation refers to a concrete implementation of a interoperability lifecycle functionality $\mathcal{F}'$.*

Looking at the smart city example, this process results in an interoperability lifecycle $\mathcal{F}'$ between the environmental aware routing system and the traffic IoT systems and the environmental monitoring system. The demand for environmental data traffic data was matched by the matchmaking function after the routing system's request for service (for example on a Smart city digital marketplace platform) . A subsequent automated negotiation resulted in two interoperability contracts with the respective terms and conditions between the routing system and each DSP. The transaction functionality involves the consumption of the traffic and environmental data by the routing system and payment for service. A dissolution could be triggered, for example when one of the providing systems become offline or a better providing system is found by a new matchmaking request from the routing system side.

5.5 The transaction optimization problem

To accomplish runtime interoperability, the interoperability module, offers parameters which can be used at runtime by the AC module of the I-IOP agent to adapt the transaction functionality towards an interoperability objective. This is essential in order to address the runtime nature of interoperability, since only at runtime the composition of systems in the ecosystem is known, after they were discovered and an interoperability contract has been settled. This autonomous optimization step is missing from standard interoperability mechanisms which only rely on design-time analysis and assumptions by developers on design-time integration aspects. They fail, as soon as any of these assumptions are not met anymore.

Therefore, achieving runtime interoperability between multiple IoT systems requires, besides establishing an interoperability lifecycle instance, also a process of solving an **objective function** $\mathcal{O}(\mathcal{F}(\mathcal{T})')$. Checking whether the objective of interoperability is achieved or not corresponds to verifying whether the description of some state of interoperation $ contains the desired effects.

"When a SLA is negotiated, the client's objective is to persuade the service provider to provide it with the best possible utility. The client's utility can be expressed as a function of throughput and price" [Wil09]. Since the interoperability contract as the outcome of the negotiation functionality is similar to a SLA the *transaction objective* can be described as optimizing the semantic and pragmatic processes in order to be in line with the negotiated interoperability contract terms. To solve the transaction optimization problem it is necessary to reach an optimal joint policy of interaction and communication π between a subset of IoT systems of the IoT ecosystem. The joint policy can be split into the policy of the *DSC* system(s) $\pi_{\mathbb{C}}$, the policy of the $INSP$ system and the policy of the *DSP* system(s) $\pi_{\mathbb{P}}$.

Modeling the transaction optimization problem in IoT ecosystems will be based on the framework of **Markov decision problems (MDP)** [OA16] .

Definition 5.19 (MDP) *A markov decision process (MDP) is a 5-tuple* $\langle S, A, \tau, r, \gamma \rangle$, *where*

- S *is the set of possible states*
- A *is the set of possible actions*
- τ *is the state transition function* $\tau : SxA \longrightarrow S$
- r *is the reward function, defined as* $r : SxA \longrightarrow S$
- γ *is a discount factor* $\gamma \in [0, 1)$

A MDP models a *"discrete time planning task of a single agent in a stochastically changing environment, on the condition that the agent can observe the state of the environment"* [OA16]. Markov decision processes are widely used to model controlled dynamical systems in control theory, operations research and artificial intelligence [SB17]. A MDP models a set of possible states, actions and a state transition function which translates a state-action pair into the next state. After each state transition, a reward function produces a reward for each agent, based on the value of the new state. A discount factor determines *"the relative importance of future rewards"*, which means, how much a reward contributes to the learning objective of the agent [TMS17].
The following assumptions are taken in order to model the interoperability between IoT systems as a sequential transactions process using Markov decision processes:

1. Reaching an optimal state of interoperability can not be assumed to be a one-step process. Rather it is a continuous process which adapts to the requirements at runtime and is subject to change when the context of interoperability changes.
2. It can not be assumed, that if a state of interoperability is reached at one time instance, that it will remain steady forever. Thus, a constant monitoring of the interoperation is necessary in case of disruptions. Interoperability is not "all-or-nothing" but rather a process which leads to a more or less desirable outcome for all participants.

Thus, by applying the MDP based model to the problem of optimizing interoperability in IoT

ecosystems, IoT systems are enabled to be autonomous and can deal with changes at runtime without a designer needing to foresee all potential possibilities in advance [KH18]. To restrict complexity, only **deterministic MDPs** will be considered in which the reward functions r and the state transition function τ are not subject to change at runtime. While in case of a centralized interoperability lifecycle implementation, the MDP refers to a single-agent problem, in the case of a **decentralized implementation**, it becomes a multi-agent problem since the optimization process is decentralized. Thus, in order to formalize the interoperability optimization problem in IoT ecosystems for decentralized interoperability functionalities, we need to extend MDPs to **multi-agent decision problems (MADP)** where IoT systems operate inside a markovian multi-agent environment (MME) [OA16]. MADP are derived from Markov decision problems in a multi-agent setting, more specifically from decentralized partially observable markov decision problems (DEC-POMDP). The difference between a DEC-POMDP and a MDP is that the execution phase is decentralized and that the agents only have incomplete information about the environment: each agent may only use its own observations to select its actions. So instead of directly observing the state, the agents receive a finite set of (joint) observations and are given a finite set of (joint) actions to change the state of the environment. As such, DEC-POMDPs allow for decision making in multi-agent settings which makes it a fitting model for the coordination optimization problem in IoT ecosystems since it is is a rich framework to formulate sequential decision making and control problems for a distributed group of agents collaborating to achieve a common goal under uncertainty [Sin18]. An formal definition of a MDP and a DEC-POMDP is given by [OA16].

Definition 5.20 (DEC-POMDP definition by [OA16]) *A decentralized partially observable Markov decision process (DEC-POMDP) is defined as a tuple* $\mathcal{M}_{DecP} = \langle \mathbf{D}, \mathbf{S}, \mathbf{A}, T, R, \mathbf{O}, O, h, b_0 \rangle$, *where*

- $\mathbf{D} = \{1, ..., n\}$ *is the set of n agents*
- S *is a finite set of states*
- A *is a finite set of joint actions*
- T *is the transition probability function*
- R *is the immediate reward function*
- $\mathbf{O}$ *is the finite set of joint observations*
- O *is the observation probability function*
- h *is the horizon of the problem*
- b_0 *is the initial state distribution at stage* $t = 0$

Agent communication In a decentralized interoperability functionality implementation, it is important to define a mechanism for agents so that the communicated messages between the agents affect their internal states (beliefs) (and thus the state of the interoperation) in such a way, that it benefits in solving the interoperability problem, as described previously. In this model, similar to [PL05], communication is defined as the process of changing the (belief-) state of a particular agent so that other agents can perceive the modification and decode information from it. This is in line with definition 5.1. In this model, successful communication has to be seen as a process that aims to realize a condition of mutual knowledge relative to a communicative intention [Bar11]. Communication is seen as a social activity of a combined effort of a least two participants, who consciously and intentionally cooperate to construct together the meaning of their interaction. The question is: **what** information agents communicate in order to achieve this.

To answer this question, we consider the case in which we assume a centralized communication scenario with a communication channel between each IoT system. In this particular case, all agents receive the **full** state information(as state observations) as an aggregation over all the individual

state observations of each respective IoT system $\$_A, \$_B$. When agents communicate for example their local belief states to other agents, this will change the internal belief state of the other agent (i.e. his information base increases). Since the agent now has more knowledge about the environment, his subsequent actions will be influenced by this knowledge, resulting in a change of the state of interoperability. This extends the DEC-POMDP model into a DEC-POMDP-COM:

Definition 5.21 (DEC-POMDP-COM definition by [OA16]) *A decentralized partially observable Markov decision process with communication (DEC-POMDP-COM) is defined as a tuple* $\langle \mathcal{M}_{DecP}, \epsilon, C_\epsilon \rangle$, *where*

- $\mathcal{M}_{DecP}$ *is a DEC – POMDP,*
- ϵ *is the alphabet of possible messages that the agent can send to other agents ,*
- C_ϵ *is the communication cost function that indicates the cost of each possible message.*

In a DEC-POMDP-COM under instantaneous, noise-free and cost-free communication, a joint communication policy that shares the local observations at each stage is optimal [PT02]. However, in the practical case, we cannot rely on cost-free communication. This fact is accommodated through an additional cost function C_ϵ which defines the cost for each message the agents communicate. Also, the internal state of a particular agent might contain private information regarding the system logic which should not be made public to other agents. The state could for example contain sensitive user information, such as his current geo-location. The agent thus needs to be free to decide, how much of its state observations the subset contains and is potentially limited in this regards by laws and regulations. This is achieved by the agent's mechanism for auxiliary observations which is part of the agent model. It refers to the process of how information is communicated between the agents.

Constraints The environment of a DEC-POMDP or a MDP, can either be a single-step or multi-step environment. In a single-step environment, feedback for actions is returned after each step, whereas in a multi-step environment a chain of actions is required to receive feedback [Ste17]. In the runtime interoperability case, we restrict to single-step problems since it is assumed that the IoT system is able to observe the effect of its action immediately after each action. Furthermore, multi-step actions are very complex to analyze and should thus only be considered after an initial

single-step model has been evaluated.

Although interaction in an IoT ecosystem is of course not time-restricted and should be considered an infinite process, we will restrict ourselves to the model of finite DEC-POMDP systems due to the computational traceability. Also, the application of the DEC-POMDP will be limited to the special case, where not the individual, but the joint observation identifies the true state. In other words, if the observations of all the agents are combined, the state of the environment is known exactly. This is referred to as jointly or *"collectively observable"* [OA16] and can be described as a de-centralized Markov decision process (DEC-MDP). Since, however in this model, the agent itself still has an incomplete view on the environment, we need to introduce a communication method between the agents. To constrain the complexity of the problem further, we will only consider collaborative settings, i.e. all IoT systems get a positive reward for improving interoperability [OA16], which means that they are always rational acting in a sense that they try to improve the interoperability. However this does not mean, that they all receive the **same** reward, since this always ties to the customer utility.

5.5.1 Optimization criteria

The problem we still face is a proper formalization of the actual **goal** the agents are supposed to achieve, i.e. **what** they are supposed to optimize in the $\mathcal{O}_{\mathcal{T}}$ objective. The goal in the case of a MDP is defined as the **optimization criteria**. An optimization criterion lays out exactly what the optimization process is supposed to optimize [OA16]. Since we are dealing with single-step problems in the interoperability optimization problem, we can define the following optimization criterion for runtime interoperability:

> **Definition 5.22 (Transaction optimization criterion)** *The transaction optimization criterion is the maximization of the accumulated interoperability reward r of an I-IOP agent at each timestep t of a transaction T.*

The optimal joint policy of a list of IoT systems $pi^*(\sigma_t) = argmax_{a \in A} r(\sigma_t, a)$ is the policy which always chooses the action that maximizes the reward, given the current situation description σ_t [Ste17]. The *reward* is defined by the *utility function* of an I-IOP agent [Ste17].

In the case of the transaction optimization criterion the utility of an I-IOP agent component is to jointly maximize the semantic and pragmatic interoperability between the IoT systems. We can

now define the transaction optimization problem as follows:

Definition 5.23 (Transaction optimization problem) *A transaction optimization problem for a number of I-IOP agents can be described as a DEC − POMDP − COM which consists of the following elements:*

- I_{OC} *which is the transaction optimization criterion*
- G *is a set of I-IOP agents*
- C *is a cost function, i.e. a function that specifies for each I-IOP agent a mapping between action and costs*
- *all the other elements that are specified in the DEC − POMDP − COM model*

The task for an I-IOP agent $i \in \mathbb{G}$ is to optimize I_{OC} by applying a series of actions $\{A_1, ..., A_N\} \in A_i$ to affect the interoperability information states $\mathbb{I}_i$ with other agents $j \in \mathbb{G}$. The task for the I-IOP agent designer is now to specify the representation of $\mathbb{I}_i$, the information state update function for the I-IOP agent τ_i and the action selection policy π_i to maximize I_{OC} respecting the cost for actions C. In the next chapter we will see how this is practically done and also, how the I-IOP agent can autonomously learn π_i.

5.6 Quantification of the interoperability state

Referring back to the related work section, existing metrics for quantifying interoperability have been reviewed. The literature shows, that quantitative evaluations are rare. In order to provide runtime interoperability between IoT systems however, a quantitative, measurable analysis of the interoperability state between the systems is essential. Since interoperability, on the technical level, mainly revolves around semantic and pragmatic factors, two metrics are introduced in this section: the **semantic and pragmatic interoperability (IOP) metric**. The LCIM metric is used as a reference metric since it presents the most practical approach to measure the state of interoperability. To briefly recapitulate, semantic and pragmatic interoperability refer to the **meaning** and **use** of data in a communication. Based on these two concepts a utility function framework is defined, which is then filled by customer and developer requirements at runtime to provide the utility function.

Semantic interoperability metric Semantic interoperability refers to a shared meaning of message content inside transactions between the IoT systems. Since IoT systems use different descriptors in order to describe their data, ontologies are needed which objectively describe the content of messages. IoT systems need to use semantic processes to enrich the message content with ontological information in order to be semantically interoperable. Of course this raises a further interoperability problem, since there needs to be a common agreement on the ontology being used. However this issue can be solved inside the interoperability lifecycle in the second phase ($\mathcal{F}_{\mathcal{N}}$), where the IoT systems agree on a common ontology as part of the contract settlement. Ontology alignment (or matching) algorithms could furthermore be used to help align concepts from different ontologies.

> **Definition 5.24 (Semantic process)** *A semantic (interpretation/translation) process is the function $\mathcal{F}_{sem}(\beta, \mu_i)$ of an IoT system i which receives a message input μ_i and produces a semantic embedding of μ_i in ontology β .*

Since semantic interoperability, according to the LCIM model, requires syntactic interoperability, the syntactic interoperability measure is embedded in the semantic interoperability metric.

Definition 5.25 (Semantic interoperability metric) *The semantic interoperability metric measures the accumulated embedding score over the elements of a transaction. More specifically, it is measured on each side of an IoT transaction by the respective IoT system as the semantic interoperability gap IOP_{G_S} and contains the following measures:*

- *Protocol mismatch $\Cap$: A protocol mismatch occurs when the receiving and sending IoT system use different interaction protocols*
- *Semantic mismatch Γ: A semantic mismatch occurs when the sender and receiver use different ontologies than negotiated.*
- *Validity τ: A message $\beta(\mu_i)$ is valid if it contains all elements that were negotiated in the interoperability contract required for proper processing the message.*

Note that the measures can be both discrete and continuous, for example a semantic mismatch can be measured through a binary measure or through the amount of mismatch in ontological terms.

We can then formulate the *semantic interoperability optimization* criterion (as one part of the transaction optimization criteria) as follows:

Definition 5.26 (Semantic interoperability optimization criterion) *The semantic interoperability optimization criterion maximizes the semantic interoperablity metric between two IoT systems.*

Based on this criteria, the concept of **semantic interoperability** is redefined as:

Definition 5.27 (Semantic interoperability) *Two IoT systems $\mathbb{C}$ and $\mathbb{P}$ are semantic interoperable when their semantic processes $\mathcal{F}_{sem,C}$ and $\mathcal{F}_{sem,P}$ maximize the semantic interoperability criterion.*

Pragmatic interoperability metric Messages are sent between systems in order to change the systems state, that is, messages are always sent with some intention. The pragmatic interoperability problems arise when the intended effect of a message differs from the actual effect of

the message. Pragmatic interoperability can be seen as a particular *state* of interoperability between IoT systems where the system's expectations about the intended outcome (or desired state) of an interaction match. Interoperability is achieved at this level when processes serving different purposes under different contexts by different information systems can be composed to jointly support a common intention [LLL14]. In this thesis, the term **purpose** is used equivalently with **goals**. For example, the purpose of a smart parking system is to assist a driver in finding parking spots in a city while the goal of a routing service might be to calculate the shortest route from point A to point B. By composing these systems, their goals need to be aligned properly. The abstract goals have specific requirements associated with them, concerning quality / precision and timeliness of results. Until now there is no universal metric to quantify pragmatic interoperability. In order to quantify it between different IoT systems, a straightforward way would be to evaluate the business logic of the underlying IoT systems and assess it similar to the semantic interoperability metric through matching and translating/interpreting on the business process level. Business processes can be modeled through ontologies as well, which means, the pragmatic interoperability problem could be reduced to a semantic interoperability problem. To be more specific, a business process matching tool could be used which translates between different, semantically annotated business processes and measures (analogous to the semantic interoperability metric) the level of pragmatic interoperability. This would correspond to an *a-priori* measure of pragmatic interoperability which defines, how well the processes are aligned. However, this is only the theoretical, ideal case. In reality, such a process is difficult to establish since DSP (or DSC systems) might not want disclose their business logic processing (due to intellectual property (IP) rights issues, to prevent competitors to analyze their offerings). Also it requires the developer of the DSP (or DSC system) to first of all define such a business process model. It is not realistic to generally assume that a DSP system will advertise its business process for pragmatic interoperability analysis. Another way to measure pragmatic interoperability is *a-posteriori*, i.e. during runtime, after the interaction between the systems has already taken place. This procedure has two advantages: (i) Systems do not need to disclose their business logic a-priori the interoperation, (ii) the overhead for semantic annotations (especially on the complex business logic side) is reduced.

The *pragmatic interoperability metric* is defined to measure the **efficiency** of reaching a desired functional state of interoperation between multiple IoT systems, where efficiency refers to the match between requester-intended state and the subsequent state, after the request has been processed. Note that such a definition is aligned with an MDP process, as defined earlier. This means,

the intended state can be compared with the actual state of the environment, after a transaction was processed and a new state of interoperation has emerged on a quantifiable basis. Moreover, the pragmatic interoperability metric measures any mismatch with regards to the non-functional constraints of the interaction between consumer and provider which were settled in the interoperability contract. Bear in mind, that terms which were not settled in the interoperability contract can still be compared however usually it is expected that systems settle the most important interoperability issues in a contract (but, referring back to the case about intellectual property, any information regarding this topic which might compromise a consumer or provider's IP, will also not be reflected in the contract).

Definition 5.28 (Pragmatic interoperability metric) *The pragmatic interoperability metric quantifies the reactions of IoT systems to exchanged transactions and the customer requirements for the digital service. It is measured as the pragmatic interoperability gap IOP_{G_P}, i.e. the difference between the current state of interoperability IOP_S and the desired state by the customer of the digital service IOP_D : $IOP_{G_P} = IOP_D - IOP_S$.*

Although this metric alleviates the *a-priori - a-posteriori* dilemma, the quantification problem is still not solved since it is still open, how to quantify the current state of interoperability and the desired state to calculate IOP_G. For this we require, what is referred to as a *pragmatic process*.

Definition 5.29 (Pragmatic process) *A pragmatic process is the function $\mathcal{F}_{prag}$ of an IoT system which receives a semantically embedded message σ and a customer (environment) context C and produces a reaction $pr(\sigma, C)$ that changes the state of the environment* $\mathbb{S}$.

The pragmatic process is considered a *black-box*, i.e the functionality is hidden however its externalities can be quantified. A suitable metric to do this can be developed using functional and non-functional IoT service aspects . For example, a maximum message delay might be specified as part of the negotiated interoperability contract in step $\mathcal{F}_{\mathcal{N}}$ and measured during execution. A pragmatic interoperability mismatch then occurs if the negotiated and actual message delay deviate. Another example could be the desire of a customer of a smart home digital service to provide a functionality to dim the lights. A pragmatic interoperability mismatch would occur, if the consuming system of such a service (e.g. a smartphone interface) would issue a request to

dim the lights inside the living room, while the service actually turns on the lights in the bedroom. Obviously, the metrics that can be defined here tend to be quite use case specific, making it difficult to build a general purpose solution. Thus any use case specifics are abstracted as the current and desired state IOP_D and IOP_G quantified as:

- efficiency: how efficient the state change was, where efficiency is measured as a customer defined function of the current and previous state
- accuracy: how accurate the resulting state resembles the desired state
- availability: The availability of the provider(s)
- execution time: The time it takes to reach the desired environment change
- cost: The cost for changing the environment state

One can distinguish these properties into three groups, similar to those defined by [Che09]: (i) cost, (ii) delay and (iii) quality

Analogous to the redefinition of semantic interoperability, pragmatic interoperability in the context of the pragmatic interoperability optimization criteria can be defined as follows:

Definition 5.30 (Pragmatic interoperability optimization criterion) *The pragmatic interoperability optimization criterion maximizes the pragmatic interoperability metric between two IoT systems.*

Definition 5.31 (Pragmatic interoperability) *Two IoT systems* $\mathbb{C}$ *and* $\mathbb{P}$ *are pragmatic interoperable if their pragmatic processes maximize the pragmatic interoperability criterion.*

We can now define the combined reward measure for semantic and pragmatic interoperability.

Definition 5.32 (Reward measure) *Consider a transaction* Λ *between two IoT systems* $\mathbb{C}$ *and* $\mathbb{P}$ *and a current state of interoperation* $\mathbb{S}$*. The transaction leads to a particular state change of interoperation* $\mathbb{S}_{t+1}$ *which is the outcome of the semantic & pragmatic processes of both IoT systems* $\mathcal{F}_{sem}, \mathcal{F}_{prag}$*. The outcome of the processes produce the the interoperability gap* $IOP_G = (IOP_{G_S}, IOP_{G_P})$ *which is used as the reward measure for the I-IOP agents.*

5.7 Related work

Based on the runtime interoperability requirements and the previous model for runtime interoperability from the previous section, we can now compare different software based interoperability solutions (see chapter 3) with respect to their capability to satisfy the runtime interoperability requirements and fit to the model of runtime interoperability. The following table lists the most relevant approaches in this respect one per row, with one column per requirement. A + sign means, that this approach supports a specific requirement, while – means, that it does not. A ? refers to unknown information about the requirement support.

Requirement/Approach	CONNECT[GIB12]	HYDRA[ERA10]	MUSIC[RBD09]	CHOReOS[BHKL13]	UBIWARE[NKK+09]
Measurability	-	?	?	?	?
Discoverability	+	+	+	+	+
Service orientation	+	+	+	+	-
Semantic mechanisms	+	+	+	+	+
Pragmatic mechanisms	+	-	+	-	-
Adaptivity	-	-	+	+	+
Learning behaviour	-	-	-	-	-
Autonomy	-	?	-	-	+
Interoperability reasoning	+	-	+	-	+

Table 5.1: Mapping of runtime interoperability requirements to existing interoperability solutions in the IoT space.

A sophisticated approach toward interoperability between web services was developed in the CONNECT project [GIB12]. They propose an emergent middleware approach that facilitates dynamic interoperability by forming an interoperability layer to connect two systems at runtime. Compared to static middleware solutions, the project addresses the problem of spontaneous interactions as a characteristic for future distributed systems (such as the IoT), when systems do not adhere to the same standards[VF13]. They create a solution for message interoperability, i.e. the ability to interpret messages and behavioral interoperability, i.e. the ability to mediate interaction protocols at runtime[SBC+15]. Dynamic approaches such as proposed in the CONNECT project, provide maximal flexibility, however they are resource intensive and not suited for real time applications. Although there are currently adaptive middleware approaches, the adaptation logic is still specified at design time and not context-aware [RMjP15]. Service discovery and composition, inherent to the proposed middleware based approaches for IoT, need to be autonomous dynamically

adaptable [RMjP15]. If we compare this outcome with the interoperability requirements proposed by [Gué14], we conclude that there are still solutions missing in order to properly deal with the nature of ultra-large scale systems such as the IoT. If a system must adapt its operation due to changes in service dependencies, an adaptive middleware solution that is build on predefined policies and rules will only work, as long as the potential changes can be anticipated at design-time. In order to address runtime interoperability, these changes must also work for system changes that were not anticipated. This requires to place more intelligence on the system's side, so that the system itself can decide about necessary changes at runtime. [GIB12] do not define an interoperability metric which can be used autonomously by systems at runtime in order to asses the current level of interoperability. Furthermore, the approach does not feature a feedback mechanism thus the adaptability to changes at runtime remains questionable and not answered. Thus, autonomous system behaviour cannot be fully expected .

The other approaches mostly confirm, that semantics and service orientation have been the predominant approach to interoperability in the IoT, with all approaches supporting discoverability, service orientation and some kind of semantic mechanism and ontologies. Also adaptivity is addressed by most of the approaches in some way, although this usually restricts to fixed rule-base policies. The UBIWARE approach focuses on a multi-agent based approach that addresses autonomy and proactiveness. But interoperability between different discovery protocols is not yet supported [RMjP15]. The MUSIC approach presents another sophisticated IoT middleware that consists of a self-adaptive architecture and a context manager. However, it does not support system autonomy, since the MUSIC platform handles all interoperability functionality, acting as a centralized system. Also, learning is not supported in this approach as well as no information about a measurability metric for interoperability is given. HYDRA follows a service-oriented model and also supports dynamic reconfiguration, self-configuration and semantic interoperability [RMjP15]. CHOReOS addresses the semantic and synactic interoperability levels and allows large scale IoT service choreographies [RMjP15] but fails to support higher-level, dynamic interoperability requirements and real-time support [RMjP15]. Overall, we observe an absence from all (or most) current approaches to address the measurability, autonomy(except UBIWARE) and learning behaviour requirements for runtime interoperability.

5.8 Summary

This chapter introduced a novel conceptualization of interoperability inside the framework of IoT cross-platform ecosystems - referred to as *runtime interoperability*. First of all, the necessity to consider interoperability as a runtime problem compared to usual design-time approaches, which are not automatically adaptable at runtime, was introduced. Derived from this, requirements were gathered for the design of IoT systems in order to provide runtime interoperability. A theoretical model of runtime interoperability and especially the transaction optimization process, based on the theory of multi-agent decision problems, was then defined before reflecting on the related work were we observed that there is currently no solution which covers all runtime interoperability requirements. The next chapter will therefore propose a concrete architectural model and implementation of the runtime interoperability model to close this gap.

Chapter 6

An architectural approach to solve runtime interoperability

6.1 Aim

The previous chapter presented a theoretic introduction to the runtime interoperability problem in IoT ecosystems and introduced an IoT system model to solve the runtime interoperability problem (see figure 5-1). Furthermore a number of requirements for the design of IoT systems have been derived to make them per definition *"runtime interoperable"*. Until now it is still unclear, how such an IoT system model can be put into practice. This aligns with the next research question of this thesis, i.e. how IoT systems need to be designed in order to implement the runtime model of interoperability.
In this section, a concrete architecture is presented that will implement the requirements for runtime interoperability in IoT ecosystems, specifically for the two I-IOP agent components (interoperability module and AC module). The architecture description is split into two parts:

First an explanation of different models to implement the interoperability module is given (section 6.2). It is explained, how a centralized implementation differs from a decentralized one and provide technical building blocks for realizing these implementations. This is important to understand the practical implications for the I-IOP agent since each architecture brings its own benefits and issues. Next, an introduction to the architecture for the AC-module is given in section 6.3 which builds on the methods from the Organic Computing domain. It integrates a controlling

mechanism enabling I-IOP agents to gradually adapt and improve interoperability between IoT systems by solving the transaction optimization problem. The integration of both the interoperability module and the AC-module into the IoT architecture is then presented in the final section.

6.2 Interoperability module implementation

As a reminder, the task of implementing an interoperability lifecycle functionality inside the interoperability module $\mathcal{F} = (\mathcal{F}_{\mathcal{M}}, \mathcal{F}_{\mathcal{N}}, \mathcal{F}_{\mathcal{T}})$ is composed of sub-functionalities $\mathcal{F}_{\mathcal{M}}$, $\mathcal{F}_{\mathcal{N}}$ and $\mathcal{F}_{\mathcal{T}}$ as depicted in section 5.4.2. Different options ranging from a completely centralized to a completely decentralized implementation can be considered, which was already briefly introduced in section 4.3.6. This section explains how these theoretic concepts are actually implemented.

Centralized implementation The simplest model for implementing the interoperability lifecycle is through a centralized implementation, also referred to as a *broker* architecture in literature [MMP19]. Figure 6-1 presents an abstract flow diagram including the interoperability lifecycle related responsibilities.

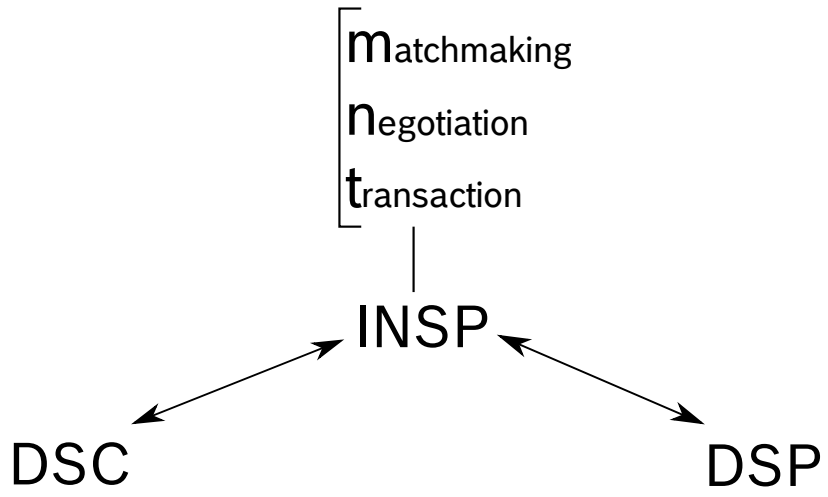


Figure 6-1: Centralized implementation of the interoperability lifecycle functionality.

In a centralized implementation, all interoperability related functionality is enforced by the Infrastructure provider (INSP) system which acts a centralized mediator. Similar to this architecture, in [BPGG11] the class of *transparent interoperability solutions* is described, where *"protocol specific messages, behaviour and data are captured by the interoperability framework and then translated to an intermediary representation"*. Examples for this kind of implementation are the smart

production and smart agriculture use cases which both feature a dedicated infrastructure service provider component acting as a centralized coordinator.

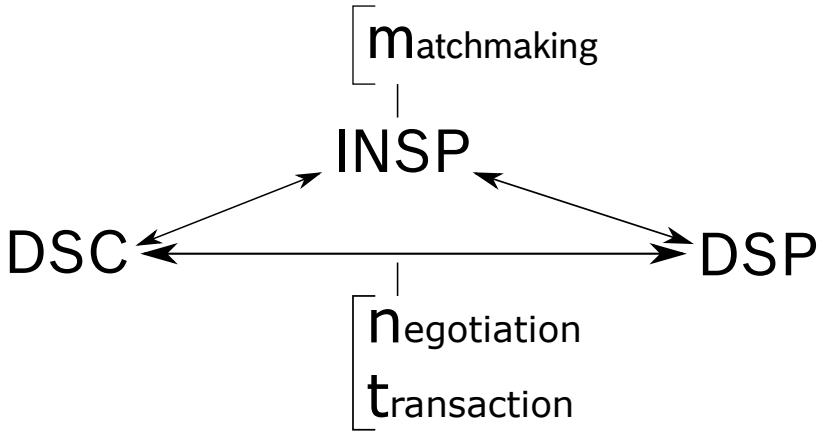


Figure 6-2: Semi-decentralized implementation of the interoperability lifecycle functionality.

Semi-decentralized implementation In a semi-decentralized implementation (figure 6-2), the INSP system still acts as a central hub for the matchmaking process, however the negotiation and transaction functionalities are distributed towards the DSC and DSP systems. Distribution here means, that the functionality for negotiation and transaction processing needs to be implemented by the DSC and DSP systems in an interoperable way. For example, in the smart production use case this could mean that negotiation on an interoperability contract between the factory management system and a production machine is not assisted by the INSP but needs to be performed by the I-IOP agents operating on the respective platforms. From an interaction flow perspective, after a matchmaking between DSC and DSP systems has occurred, any further information exchange is done peer-to-peer between them. The INSP system may however still be involved through periodically submission of transaction related data from the other systems or for dispute settlement.

Decentralized implementation The decentralized implementation differs from the semi-decentralized one by completely removing the INSP system in the interaction, i.e. the INSP system's responsibilities are distributed onto the DSC and DSP. That means, the logical central point of matchmaking, which is still left in the semi-decentralized version is decentralized to the DSC and DSP systems as well. Figure 6-3 presents the described architecture in an abstract way. In this implementation of an interoperability lifecycle, there is no concrete instantiation of an INSP system anymore. This

style of implementation suits itself particularly well to open environments with lots of changes of interoperating systems such as in the environmental-aware routing use case. Here, a decentralized implementation could well lead to easier market access for the smart city oriented IoT systems and offer better scalability and resilience.

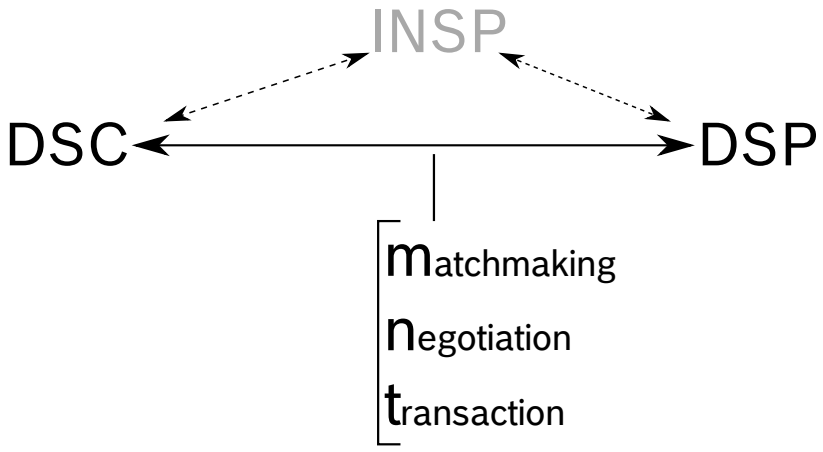


Figure 6-3: Decentralized implementation of the interoperability lifecycle functionality.

6.2.1 Matchmaking functionality implementation

The matchmaking functionality is defined in an abstract fashion in definition 5.13. To implement the matchmaking functionality - information regarding available digital service demand and supply needs to be available in the IoT ecosystem. Based on the underlying lifecycle implementation this is achieved in different ways. The beginning of this process always needs to involve technology for IoT systems to engage with each other. [MMP19] provide an extensive overview over state of the art service discovery solutions in the IoT space which revealed that few approaches actually use semantic technologies for discovery and most approaches feature a centralized directory based architecture.

In the *centralized* and *semi-decentralized* model, matchmaking (as all the other functions) is performed by the INSP agent, utilizing a central repository which features the candidates for interoperation. The agent has knowledge both about the supply and the demand for a digital service and implements a matchmaking algorithm to match DSC and DSP agents. To initiate the matchmaking, the DSC agent sends a demand request for a digital service to the INSP agent, while the DSP agent sends offer proposals. From an interoperability perspective, the DSC agent needs to be aware of the INSP agent location inside the network, and its required syntax and semantic structure for demand requests. The same applies for the DSP agent. The demand request itself usually

contains information about the required type of service, required level of service or detailed parameter settings. More details and examples for demand and supply will follow in a later section in the context of the BIG IoT architecture. After receiving offers from DSP agents, they are stored in a database by the INSP agent and the DSP agent is notified as soon as a matching DSC agent is found. Ontologies can be stored centrally by the INSP agent to be used in the matching process, presuming that demand and offer requests are semantically annotated. In this case, sophisticated semantic distance measures can be calculated to determined the matching overlap.

In the decentralized model, the matchmaking phase is more challenging to achieve since there is no central entity storing information for all IoT agents to retrieve. Thus, new IoT agents entering the ecosystem need to be broadcast their arrival to all the other IoT agents in the ecosystem which then need to possess a mechanism to store and update information about the ecosystem population. Such approaches can be found in distributed database approaches, such as distributed ledger technologies [TV07]. In a decentralized setting, the matchmaking protocol needs to be implemented jointly by all the members in the ecosystem and there needs to be a consensus mechanism on correct execution. This of course implies, that the IoT agents which enter the ecosystem possess this kind of knowledge and capabilities. Examples for these implementations can be found for instance in the Universal Plug and Play standard (UPnP)[MMP19]. In this model, DSC agents broadcast their service requests to all participants of the network who then respond with service descriptions. However, as explained by [MMP19], this architecture needs to deal with problems of excessive bandwidth and energy usage (thus especially difficult to implement with regards to constrained IoT platforms). Regarding the implementation of the actual matchmaking algorithm, an exhaustive explanation of these solutions is outside the scope of this thesis, but there is extensive literature on the topic of matchmaking protocols from the economic and Multi-agent systems (MAS) literature [Vei03]. Matchmaking algorithms range from simple crawling of service catalogs, where service providers can register their service and a crawler regularly searches for fitting request-offer patterns to match consumers with providers of services to auction based matchmaking where an optimal resource allocation is performed from an economic perspective [Vei03].

6.2.2 Negotiation functionality implementation

The negotiation functionality implementation realizes the formal negotiation functionality blueprint from definition 5.14. Negotiation is an act between IoT systems, but in a centralized setting, a specialized IoT system can perform negotiation activities on their behalf. As described in [MMP19], as

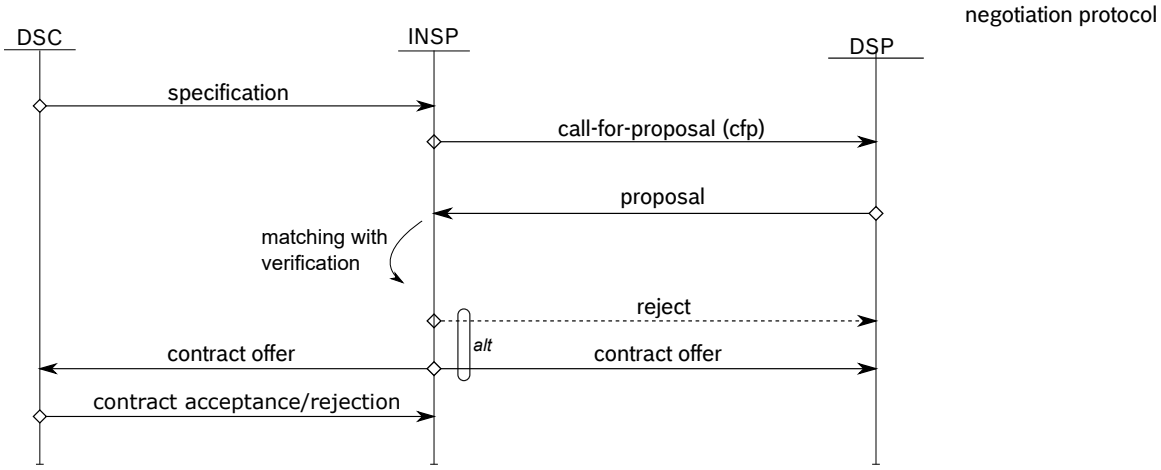


Figure 6-4: Example of a negotiation protocol for implementing the negotiation module of the interoperability lifecycle functionality.

with the example of the Cloud Sensing Brokering Platform, a centralized broker platform is used to perform negotiation and monitoring activities. In interoperability related terms, the mediator can abstract the underlying negotiation protocol details and thus also mediate between different protocols. There exist a wide range of negotiation protocols in literature, with the WS-agreement protocol being the standard in webservice oriented systems [MMP19]. The objective of the WS-Agreement specification is to define a language and a protocol for advertising the capabilities of service providers and creating agreements based on templates, and for monitoring agreement compliance at runtime [1]. Another common negotiation protocol from the agent domain is the usage of auctions, for example a Dutch auction [MMP19], or negotiation protocols from the FIPA specification[2] such as the contract net protocol. In this protocol an agent proposes a contract to another agent who then is allowed to either offer a counter-proposal or accept the offer. On acceptance, the contract is transferred into a settlement phase.

An adaptation of the WS-agreement protocol for a central broker can be seen in figure 6-4. The advantage of the WS-agreement protocol is, that it is closely aligned with service oriented systems such as the IoT digital service ecosystem. However it is also more complex to implement, and not as lightweight as the FIPA based protocols.

To implement this protocol, in a centralized ecosystem, the interoperability module's negotiation module needs to be implemented for the client and the mediator side. On the mediator side the module implements the verification matching process. On the client side the communication with

[1] http://wsag4j.sourceforge.net/site/wsag/overview.html

[2] http://www.fipa.org/repository/aclspecs.html, accessed: 27.12.2019

the INSP agent is implemented, i.e. sending the specification, proposals and acceptance/rejection messages. Note that the communication can be realized through standardized communication languages which assures interoperability on the negotiation protocol level. In the semi-decentralized and centralized versions,since the negotiation functionality is decentralized to the individual IoT agents, the necessary protocols need to reside on the client side. The agents know each others endpoints through the previous matchmaking phase. In a semi-decentralized implementation, the result of negotiation is still stored centrally in the infrastructure as trust base.

6.2.3 Transaction functionality implementation

The transaction functionality (definition 5.16) is the most challenging phase to implement as the coordination activities within this phase are maximal complex and inoperable. There are different interaction patterns that define specific rules to coordinate the behaviours of components. Specifically due to the large non-uniformity of the IoT sector, there are usually a number of interaction patterns supported by one platform [IoT16] which makes the interoperability problem more difficult in practice.
The implementation of the transaction functionality realizes the transaction flow as outline in figure 4-4.

In the centralized implementation, the transaction functionality is supported by the broker which makes the problem easier from a DSC/DSP developer perspective, since he can support the semantic interoperability through ontology matching or semantic translations. Pragmatic processing of messages still resides on the DSC and DSP agent side, however also here the broker can assist, e.g. by acting as a proxy for delay critical activities. Messages are sent to the INSP agent through HTTP or other IoT suited protocols, such as MQTT or COAP [RMjP15]. From an implementation side, this is the easiest approach to provide an interoperable transaction phase since, effectively, all interoperability problems with regards to the transaction phase are offloaded to a third party INSP provider. On the flipside, this means that since all messages need to be routed through the INSP agent this raises scalability and privacy concerns. Also, if not assisted by proper parallelization, a failure of the INSP agent will cause the complete functionality to break.
In the semi-decentralized & decentralized setting, the difference with regards to the previous implementation is the decentralization of the the transaction functionality $\mathcal{F}_{\mathcal{T}}$, i.e. the semantic and pragmatic processes are handled completely on the DSC and DSC side. This means, IoT agents are responsible for proper reaction to messages, i.e. semantic translation/interpretation as well as the

	Centralized	Semi-decentralized	Decentralized
Advantages	Easy to implement; Only one agent necessary for controlling	Better scalability; Central repository for ontologies	Transparency; No central point of failure anymore
Disadvantages	Scalability; All data visible to central broker; Lock-in effect; Restricts autonomy of IoT agents; Central point of failure	Still one party centralized	Security & Privacy - in a DLT based implementation, all information is public; Scalability - all nodes must process all computations

Table 6.1: Advantages and disadvantages of centralized, semi-decentralized and decentralized interoperability lifecycle implementations.

already implemented pragmatic processes. With regards to the transaction optimization problem, in the centralized implementation, only one AC-module is necessary for adapting the interoperability process whilst in the semi-decentralized and decentralized cases, there is one AC-module per DSC and DSP agent needed.

6.2.4 Advantages & Disadvantages of a centralized implementation

According to the definition of an IoT ecosystem, it is inherently a decentralized systems. A centralized functionality, although easier to implement, in the long run restricts the desired autonomy in IoT ecosystems. The advantages of decentralized solutions are their respect of autonomy, dynamics and lack of centralized components which might fail and thus impede interoperability. However, in terms of implementation effort it is also the most demanding case since it needs to be made sure, that all agents adhere to the protocol. Table 6.1 compares the three main approaches to interoperability lifecycle implementation as to their advantages and disadvantages. Based on that analysis, it seems that the semi-decentralized implementations presents a sweet-spot at the time of writing until technology advances in decentralized technologies have solved the scalability problem.

6.3 AC module implementation

The nature of the runtime interoperability problem requires an appropriate software architecture to deal with the transaction optimization problem (as previously defined in definition 5.23). This calls for an architecture that can, on the one hand, deal with as little pre-defined knowledge as possible and on the other hand is able to acquire new knowledge constantly at runtime and use this knowledge to adapt the transaction process. This section introduces the AC-module specification of the I-IOP agent.

6.3.1 Background on Organic Computing

As described in chapter 5, IoT ecosystems can be analyzed and constructed from a DSE perspective which already hints to their nature of being self-organizing systems [LBB12]. Self-organizing systems are naturally decentralized systems, where any knowledge available is distributed across the participating agents [KK17]. Research areas with a particular focus on the development of system architectures for self-organizing systems are **Organic computing**, respectively **Autonomic computing** [KK17]. Hence these areas seem to be ideal candidates to consider for IoT agent design and thus will be described briefly in the next paragraph for further background before going into the details of the architecture proposed in this thesis.

Organic computing architectures *"An Organic computing system is a technical system which adapts dynamically to the current conditions of its environment"* [Tom15]. Organic computing analyzes models and architectures for the design of systems which support these properties. It is closely linked to the topic of autonomic computing and feedback control systems. It emphasizes that complex systems need to have self-control and self-adaptation abilities while keeping humans in the loop for monitoring and control. The topic originally emerged from a strong industry focus since in this context the complexity of systems becomes easily untraceable due to the various parts that need to function together. Organic computing essentially is different from earlier feedback-control systems as it focuses also on the agent's ability to *"adapt its own goals, plans, resources and behaviors to meet new contexts and requirements"* [KK17]. One of the most important artifacts from the Organic Computing domain is the Observer Controller Architecture (OCA) which is depicted in figure 6-5, showing essentially a reference architecture for an Organic computing system.

Although there are already a variety of Organic computing architectures developed for different SoS-relevant domains, applying this concept to interoperability related research has not

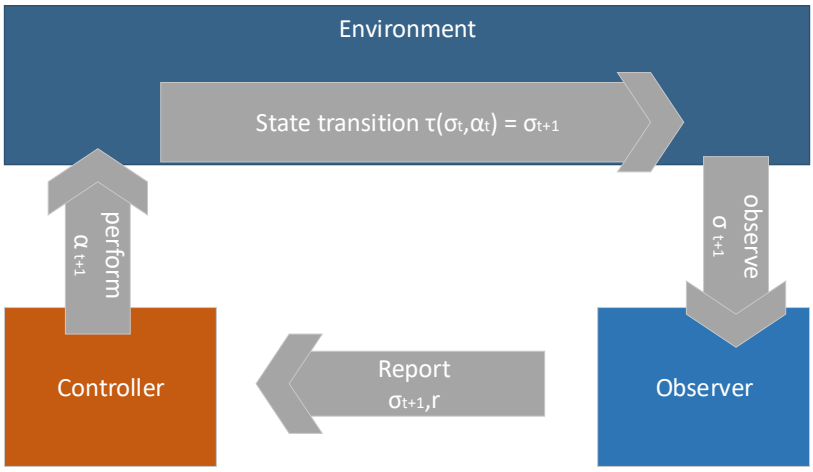


Figure 6-5: The Observer-controller reference architecture defined in [Ste17], consisting of an observer component receiving state information from the environment and a controller that performs actions to change the state of the environment.

yet been explored. Closest to this context comes the concept of *self-integrating* systems [BTW15] which describes a process in which systems or subsystems are included into an overall system and this inclusion status is continuously maintained. [TRBW16] present an approach for self-improving system integration at runtime which also features a distributed middleware with functionalities for communication, neighbour discovery and capability description of ecosystem members. The approach presented in this thesis is similar in this respect, since also for the matchmaking phase, service descriptions are required by the service providers. However, here the focus is on interoperability, specifically semantic and pragmatic interoperability, between the ecosystem entities which could be considered an extension of the capabilities of the communication platform as described in [TRBW16]. This means, in the context of self-integration this thesis specifically focus on the complete lifecycle of interoperability which entails not only neighbor discovery but also contract negotiation and transaction monitoring.

Organic computing systems are characterized by being built out of systems with properties such as *"self-organizing"*, *"self-configuring"*, *"self-healing"*, *"self-protecting"*, *"self-explaining"*, and *"context-aware"* [KK17]. These properties are usually subsumed by the concepts of *"self-awareness"* and *"self-adaptivity"* which will be explained in the following.

Self-aware & self-adaptive systems Self-aware system research is inspired from psychology and cognitive science [LCP+11]. It concerns the proactive collection and representation of knowl-

edge about a system, by that system, in order to perform intelligent reasoning on this knowledge [LCP+11]. Self-aware systems can address the issues that will govern the future deployments of IoT systems, that are: faulty and unreliable components, increasing levels of dynamics of distributed systems, the position, functionality and availability of resources during runtime and diverging requirements of future application domains [LPR+16]. For the interested reader, a thorough introduction into self-aware systems is given in [KK17], however it is beyond the scope of this section to go into that amount of detail.

Definition 1 (Definition of self-aware computing systems by [KK17]) *Self-aware computing systems are computing systems that:*

1. *learn models capturing knowledge about themselves and their environment (such as their structure, design, state, possible actions, and runtime behavior) on an ongoing basis and*

2. *reason using the models (e.g., predict, analyze, consider, and plan) enabling them to act based on their knowledge and reasoning (e.g., explore, explain, report, suggest, self-adapt, or impact their environment) in accordance with higher-level goals,which may also be subject to change. [KK17]*

According to [KK17], there are five design properties that should be considered when engineering self-aware systems. They are:

- introspective, i.e. they can observe and optimise their own behaviour (SADP-1)
- adaptive, i.e. they can adapt to changing needs of applications running on them (SADP-2)
- self-healing, i.e. they can take corrective action if faults appear whilst monitoring resources (SADP-3)
- goal oriented, i.e. they attempt to meet user application goals, and (SADP-4)
- approximate, i.e. they can automatically choose the level of precision needed for a task to be accomplished (SADP-5)

A reference architecture for self-aware systems is presented in [LPR+16] which contains the following components: internal & external sensors and actuators, self-awareness component and a self-expression component. This high-level architecture can be connected to the general Observer-controller architecture (figure 6-5). Research on self-aware systems has many intersection with

other disciplines such as Artificial intelligence, Service-based systems, Models@runtime or context-awareness [KK17]. Self-aware systems put emphasis on online learning algorithms that are used in order to gather the right information about the systems internal and external environment's state at runtime. It should be emphasized though that it is not required that all self-aware systems use the same learning strategies, rather the opposite: Heterogeneous strategies can lead to increased performance in heterogeneous environments [LPR+16] . Self-aware system research further assumes, that the self-aware behaviour is not externally fitted to the system but instead it is built into the system, i.e. are internally considered during the design of the system [LPR+16].
Self-awareness is furthermore closely linked to self-adaptivity. Self-adaptivity builds on the properties of self-awareness and context-awareness, with specific self-adaptivity concepts such as "self-configurability", "self-optimization", "self-healing" and "self-protection" in between [KMV+15]. In other words, for self-adaptive behavior, basic system properties of self-awareness and context-awareness are necessary, such as the ability of a system to monitor its resources, state and behaviour. Thus, one can view self-awareness as a prerequisite to form adaptive behavior. *"A self-adaptive system is able to automatically modify itself in response to changes in its operating environment"* [KMV+15]. Modifications in this case refer to changes in parameter settings or internal structure or other artifacts of the system [KMV+15]. A self-adaptive system consists of a managed resource and the adaptation logic. An example for the adaptation logic would be the MAPE-cycle from Autonomic Computing [LPR+16]. Adaptation is seen as the process through which a system *"becomes better suited to its environment given its purpose [...]"*[LPR+16]. A self-adaptive system distinguishes itself from any other system by its ability to adjust its behavior based on response to its perception of the environment and the underlying system state itself [KK17].

Self-awareness is considered as an important basis for IoT agents in order to be aware of their surroundings in the ecosystem and also its own goals and purpose. This serves as a prerequisite for autonomy and interoperability reasoning, two of the runtime interoperability requirements defined in section 5.3. Still, self-awareness alone is not enough. IoT agents also require the ability to take actions based on the learned knowledge from the environment. It is here that *self-adaptiveness* comes into play. Building self-adaptive systems for IoT ecosystems is especially challenging since it constitutes a SoS and the inherent uncertainty of SoS is still an active research area with regards to self-adaptation due to the additional complexity of decentralization [WA13]. For example, in [WA13] three different styles for the design of a self-adaptive SoS are presented however a general

guideline for the design inside IoT-ecosystems cannot be derived that easily. Thus within the I-IOP agent model, this open problem is addressed by considering the composition of IoT ecosystems as systems-of-systems, i.e. the IoT systems can self-adapt inside the boundaries of the interoperability lifecycle and towards the transaction optimization criteria. By integrating the I-IOP agents in the interoperability lifecycle implementations, this allows to consider different deployments of self-adaptation logic which is described by [KMV+15] as centralized, decentralized or hybrid adaptation logic and thus extending the research by [WA13] and others.

6.3.2 Runtime interoperability and self-adaptive IoT systems

Referring back to the introduction and challenges of runtime interoperability in section 5.2, the question if self-aware or self-adaptive systems are able to help in this respect and in what way needs to be addressed? Since the development of digital service ecosystems is (due to the desired properties of the ecosystem) deeply entangled with Autonomic computing (and thus also Organic computing) research, [AO16] have provided a recent overview of autonomic computing methods in digital service ecosystem design. The main outcomes of the analysis are:

- Reflexivity as a technique to support evolution of the ecosystem is completely missing from current interoperability research
- Service and pragmatic interoperability has not yet been considered by AC methods in digital service ecosystems

In an abstract fashion, a couple of key issues are needed to be overcome during the design of an AC module for runtime interoperability: (i) the problem of defining a controlling element, (ii) the problem of what to observe and how to measure it and (iii) the integration with existing IoT systems and deployment.

The problem of defining a controlling element The controller is the crucial element of the AC module since it influences the interoperability module and thus the interoperability state with other IoT systems. An essential element for the runtime interoperability performance is thus a proper design of the optimization target, or reward function. This is a well-known problem in Reinforcement Learning research, with a dedicated discipline called *Reward engineering*. During the specification of the initial AC module architecture, the following concerns have been identified that need to be addressed by an IoT developer when designing the optimization target:

- **Guidance**: The IoT developer/designer needs to be given proper guidance in designing the policy function. It cannot be assumed that an IoT developer is familiar with machine learning concepts, thus a design framework needs to elaborate and ease the design process.
- **Completeness**: The reward needs to be designed over all key outcomes that the I-IOP agent will experience. This requires a well-thought through reward function, although, even in the case that certain situations do not have an attributed reward, the agent can still be re-trained online.
- **Unambiguance**: The same reward should not be received for different actions.
- **Tradeoff between domain-specific and domain-agnostic rewards**: There is always a tradeoff with the design of the reward function between defining rewards to optimize a domain problem compared to designing rewards which are oriented towards solving the greater, interoperability problem

The problem of observability and measurability For an I-IOP agent to reach awareness of its surrounding environment, the specification of **what** to monitor is all important. Through the resulting self-awareness the agent is able to reason about the IoT system's state which is necessary as per the *interoperability reasoning* requirement. In practical terms: When an IoT system is interacting with multiple IoT services and devices, the agent is able to observe the current behavior of these components and if they affect its goal pursuit. This refers to its *internal self-awareness*. For example, in the smart agriculture use case the farm information system relies on information from multiple machines to accurately monitor the state of the farm. Since its goal is to accurately report this state to the farm operator, it is important to constantly be able to measure the state of interoperation with these systems. In case of problems it needs to be aware of its goal state and be able to perform corrective actions. As [NGC09] have pointed out, it is important for a system, if it wants to increase its interoperability with others, to be adaptive. According to the explanation provided by [Ack71], what it means for systems to be adaptive, it is evident that in order to trigger the adaptation inside the system it is important to identify, what led to the adaptation requirement. In self-aware systems, this knowledge is obtained by the self-aware components through the interoperability metrics that extract the relevant information from the raw state information. This requires them to be properly configured since this guides the adaptation process. Any misinterpretation will lead to unwanted results. Again, coming back to the smart agriculture use case,

the interoperability metrics might measure that the response time of one of the connected machines is below the required level, leading to increased latency and reduced interoperability. The farm information system uses this metric based measurement at runtime to react appropriately, e.g. by reducing the amount of requests on the machine platform or by alerting the human operator in case no automated approaches succeed.

To conclude this section, there are significant overlaps between the introduction of Organic computing systems and the desired system characteristics discussed in the previous section that are important for runtime interoperability. Thus, in the following an architecture for the AC-module based on self-aware and self-adaptive concepts is presented.

6.3.3 Architecture of the AC module

The purpose of the AC module is to adapt the interoperability module of the I-IOP agent at runtime to optimize the runtime interoperability between IoT systems. An abstract formalization of the AC module was already provided in the previous chapter in definition 5.11. The following architecture puts this formal AC module into practice, based on an Observer Controller Architecture (OCA) architecture . Observer Controller Architecture (OCA) architectures bring the advantage that they especially focuses on self-adaptivity in the context of large-scale interconnected systems which naturally fits the characteristics of IoT ecosystems, as detailed before. Also, OCA - architectures deal with emergence which naturally occurs in these complex systems since collaborations between IoT systems form and can lead to unexpected or undesired effects.

The AC-module monitors and controls the transaction phase of the interoperability lifecycle. The architecture is divided into two parts: the **observer** and **controller** parts. In figure 6-6, these parts are marked in blue and red respectively. In the general, the observer is responsible for monitoring the current state of the interoperation with respect to the interoperability contract, through *situation descriptions* which are extracted from the observed environment features alongside metrics which help to evaluate the observed situation[TPB+11], i.e. the current operational context. The actual design of the AC-module of an *I-IOP agent* is based on the generic Observer/Controller architecture (6-5) to extend IoT agents with self-aware and self-adaptive capabilities [KH19]. It is discriminated between the IoT ecosystem, which consists of the IoT systems exchanging transactions through the interoperability module of the I-IOP agent, consuming & providing digital ser-

vices tied to an interoperability contract and the AC module which implements the self-awareness and self-adaptivity functionality, to optimize the transaction module. The customer and developers provide interoperability constraints which are used by the negotiation functionality (F_N) of the interoperability module to negotiate the interoperability contract and for the interoperability metric to evaluate the state of interoperability. This achieves the self-aware design property SADP-4.

Monitor External sensors provide information for the monitor about the current state of the interoperation which originates from the data of the transactions. In this context the word *"sensors"* does not refer to IoT sensors but instead to *virtual sensors* which are able to connect to the interoperability module. The sensor can be seen as an interoperability related interface. The sensors monitor important features consisting of semantic and pragmatic information through an interface with the interoperability module. For example, they assess values regarding semantic embeddings of the last transactions alongside features of the pragmatic processes, such as efficiency, cost and transaction accuracy. The information is monitored by the observer component, either through an active information push from the interoperability module or through a pull-based mechanism which is initiated in regular intervals, for example every second. The exact method for observation and resolution is use case dependent and needs to be made configurable to suit different IoT deployments. Depending on the sophistication of the agent, more runtime-related features can be analyzed as well, for example information processing delays or use case dependent features. The interface for information exchange between the interoperability module and the AC module can be realized as a simple REST-based interface, since the semantics of the communication are relatively compact and well defined. However this detail is implementation dependent. The output of the monitor component is a fixed-size vector β which contains the state observation.

In the case of missing information from the sensor-based observations, the monitor furthermore receives data from the state estimator (SE) and the communication module (as detailed in the next paragraph). The state estimator uses the state history to estimate missing features from the direct observations which is constantly updated with new observations. For the estimation the mean over all historic values for each respective feature is calculated which makes the estimation more accurate the more experience the agent receives. Relevant information for measurability is extracted through the features extraction performed in the monitoring component. The feature

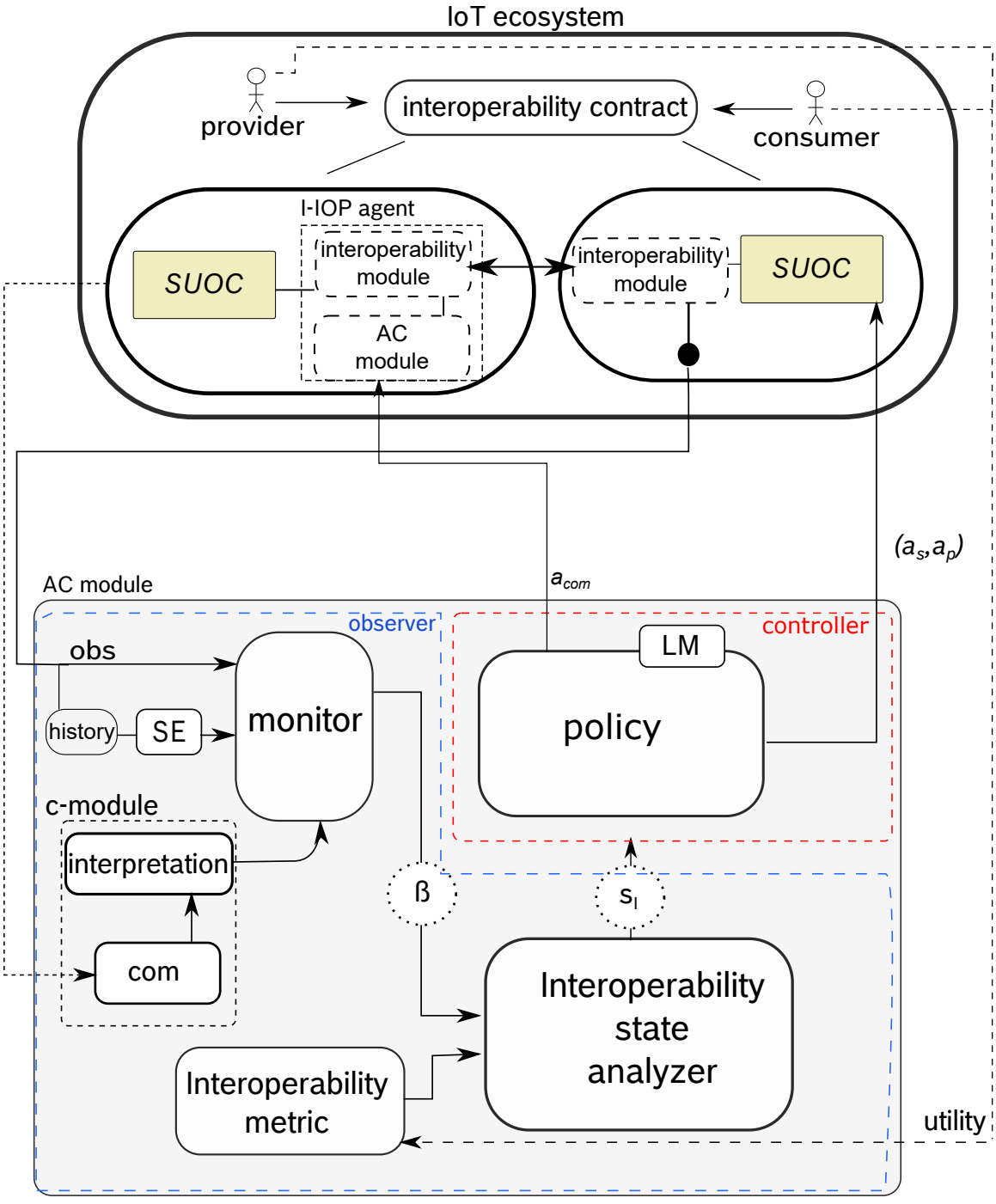


Figure 6-6: Architecture of the AC module of an I-IOP agent. The architecture is based on the generic observer-controller architecture and monitors / adapts the interoperability between IoT systems inside the IoT ecosystem.

set is designed so that the semantic and pragmatic interoperability metrics can be analyzed optimally and automatically. These two processes allow the agents to accurately measure the state of interoperability on a quantifiable basis (as opposed to the usual qualitative basis).
The monitoring component is needed to meet the SADP-1 property, i.e. to observe and optimize the behavior of the IoT agent. The actual reasoning about the interoperability state, no matter how this information was gathered, is accomplished through the monitoring component which regularly acquires information about the state of interoperability from the System under observation and control (SUOC)s and the interoperability metric for processing this information.

Communication module (c-module) In a decentralized setting, the monitoring component of the agent will not receive all information required for feature extraction, since there is no central instance through which all interoperability state related information is routed. Therefore, the communication module is used for inter-agent communication of interoperability state related features. In other words, the communication module is utilized for communication **about** the interoperation between the I-IOP agents. That means, it is used both for sending and receiving state information therefore appending an additional communication layer on top of the transaction communication layer. This requires then to also consider semantics and pragmatics on the agent communication level. However, in this case, as opposed to the general IoT ecosystem context, the pragmatics are fixed, since the purpose of the communication is defined a-priori. Also the semantics are simpler and can be fixed a-priori. Furthermore, there is an abundance of research already on agent communication standards, e.g. FIPA-ACL which can be used in an implementation to even allow more complex agent-to-agent communication protocols.
The content that is sent between the agents is set to be the last state observation of each respective agent as well as the last taken action. It has been proven that agents communicating inside a DEC-POMDP-COM problem space benefit most by exchanging their own last observations when the cost of communication is constant over all messages [GZ04] which is also the decision taken in this thesis. Furthermore, exchanging the last taken action is important in order to avoid oscillating adaptations.
Since the c-module also adds to the observations of the agent, it also contributes to property SADP-1.

Interoperability metric The interoperability metric is used to evaluate the current interoperability state β inside the interoperability state analyzer component. The metric consists of a feature extractor and a filter which analyzes the state feature vector for the policy module based on the desired utility. Hence, depending on the choice of metric, the same state information is analyzed differently. For instance, if the accuracy of message exchange or conciseness of the message content is of no importance to the interoperation, the same feature of the state space will not be weighted as important as in a case in which exact processing of requests is required. This could be for example the case in the smart city use case where the exact CO2 values of air quality might not be relevant for the routing service, rather a range of acceptable and non-acceptable values. This means, in reverse, that when the connected sensors of the environmental monitoring system do not return maximal accuracy, there is no deterioration of interoperability. On the other hand, in the smart production use case the exact provision of process parameters is essential so that a slight deviation here will be sensed through the interoperability metric and result in an interoperability decline with respect to the production machine platform. This underlines the fact, that the I-IOP agent operates inside a very specific context and that the interoperability metric is always chosen very specific to match the utility of the observed and controlled IoT system. The metric is provided by the IoT developer. The information for configuring the metric is given by the customer of the I-IOP agent. This is important to differentiate, since the IoT developer does not possess the exact knowledge of the use of the agent at runtime - this can only be provided by the customer/user of the system. The level of precision is chosen by the utility measure, either provided by the customer or developer and based on this, the features recorded by the monitor are processed. Since the interoperability metric is tight to the utility of the customer, it contributes to property SADP-4.

Interoperability state analyzer The interoperability state analyzer is a function which applies the interoperability metric to the derived features of the interoperation state β in order to create the state representation for the policy. This process is illustrated in more detail in figure 6-7.

Inside the function, β is first split into semantic and pragmatic features $\div_s, \div_p$. The feature extraction algorithm extracts the raw semantic and pragmatic features of interest which are then used to calculate the semantic and pragmatic interoperability scores. The scores are aggregated into a vector that represents the state feature representation S_I . This vector is used by the policy component, and by the learning module inside the policy component. The vector contains two information elements: *sem* and *prag* where *sem* represents the semantic interoperability related

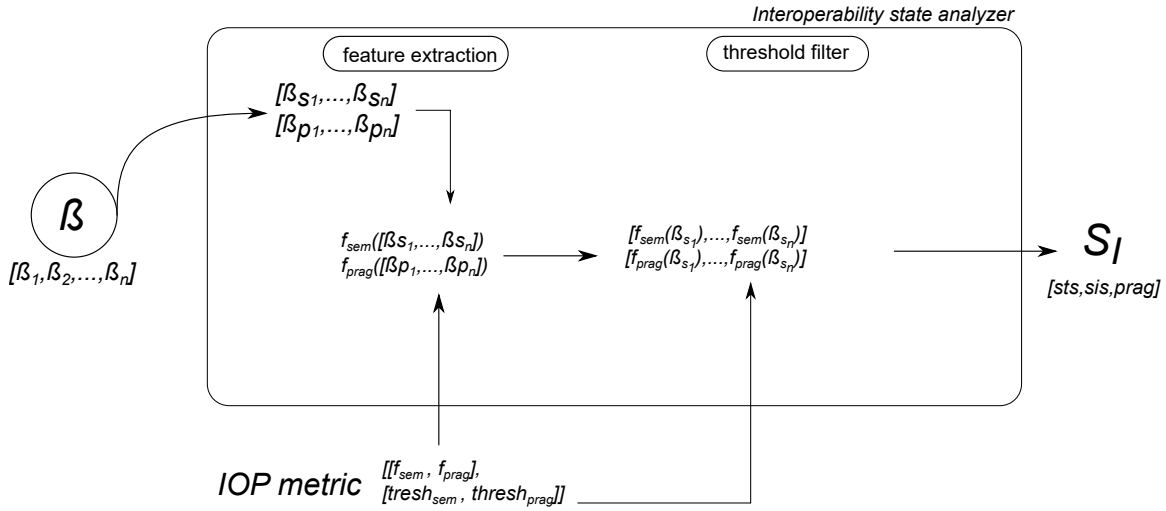


Figure 6-7: Interoperability state analysis process inside an AC module. The IOP metric is used to extract features from the interoperability state to be used by the agent policy.

content and *prag* represents the pragmatic interoperability related content. The semantic features jointly relate to syntactic and semantic interoperability related information, such as the type of protocol and ontologies used, semantic translation and interpretation information and the general validity of exchanged information. The pragmatic features concern the actual runtime information such as round-trip times for messages, cost of interoperation, and message accuracy. In general, the feature space aligns with the information contained in the interoperability contract so that the agent is able to verify if the terms and conditions of the underlying contract are met.

For the representation of the vector, a binary representation was chosen to ease the integration with the learning module. The binary representation S_I of the interoperability state can be implemented through a threshold-based system that applies multiple thresholds onto the state features which are chosen based on the interoperability metric. For example, among the multiple thresholds for the pragmatic metric are "accuracy_cutoff" or "efficiency_cutoff". If a feature lies above (or below) the threshold, it is replaced by "1" (or "0" otherwise). This representation of the internal interoperability state is optimized for the later explained LCS learning module but it also eases the design of the rule-based policy in case of an expert based manual approach. In principle, this representation can also be expanded to a continuous representation. This also brings the benefit of increased accuracy and the removal of threshold parameters which might bias the agent, however it requires to change the learning mechanism from a standard XCS to a more demanding Continuous-XCS. Furthermore, the representation could also be further compress the state information, which is an advantage in resource constrained devices. Jointly with the interoperability

metric, the interoperability state analyzer accomplishes property SADP-5.

Policy The policy π constitutes the controlling mechanism of the AC module. It is realized as a state-action mapping which maps the interoperability state description S_I into the action space of the environment which changes the transaction module operation accordingly. The actions which are the output of the policy are usually dependent on the environment and need to be provided by the interoperability module.

The policy can be either modeled manually by an expert or it is learned through a machine learning approach, more specifically a rule based machine learning methods. Rule-based machine learning uses a **learning mechanism** in order to automatically identify rules instead of relying on human judgment to handcraft all rules. A rule-based approach is preferable to other approaches since the policies are easier to interpret for humans. Rules typically take the form of an IF:THEN expression, (e.g. IF 'condition' THEN 'result', or as a more specific example, IF 'red' AND 'octagon' THEN 'stop-sign'). An individual rule is not in itself a model, since the rule is only applicable when its condition is satisfied. Therefore rule-based machine learning methods typically identify a set of rules that collectively comprise the prediction model, or the **knowledge base**. The policy representation can be either a simple table or a more complex representation, such as a neural network or decision tree, depending on the requirements. For example, in smaller,discrete state spaces a table representation is usually sufficient and easier to interpret for an expert. But, in continuous state spaces a function approximation approach, through neural networks for example, is preferable due to the well-known problem in the machine learning domain of the *"curse of dimensionality"* [SB17]. The curse of dimensionality refers to the fact, that with larger and larger state spaces, the information content increases exponentially and thus it is more and more difficult to achieve reasonable results.

The designer needs to know, when to choose a machine learning approach over manual, expert-crafted design approaches to implement the policy. Rule based systems are deterministic in nature which means, that not having the right rule in place can result in false positives and false negatives. Rule-based systems can start of quite simple, but can become rather unwieldy over time as more and more exceptions and rule changes are added. This means, in complex environments, an expert needs to be well aware of the configuration space. Naturally, the expert based approach does not align well with the idea of runtime interoperability, where the IoT agent

is supposed to learn to act autonomously in its environment. Also, if data and scenarios change faster than one can manually update the rules, machine learning approaches are more scalable and efficient. In complex scenarios, one can reach a point of losing track of the rule set easily and of how many exceptions there are. Machine Learning methods are appropriate in application settings where experts are unable to provide precise specifications for desired program behavior, but where examples of desired behavior are available, or where it is possible to assign a measure of goodness to examples of behavior. Rule-based machine learning is easier to maintain in complex environments, since the decision process itself can be considered a blackbox. Since the machine learning algorithm itself discovers the rules of the domain, it can be continuously updated and thus adapt flexibly to new environments. Still, training data or feedback mechanisms are necessary to be designed by humans which can be difficult and time consuming. One can say that rule-based machine learning focuses more on the outcomes rather then the entire decision making process, making machine learning approaches more flexible and less susceptible to some of the problems encountered with manual designed, expert based systems. In favor of manual approaches, they are in general easier to setup and do not require (potentially lengthy) training cycles. This makes them preferable in simple domains with only a limited set of required rule-action sets.

Learning module For the learning module, a suitable machine learning mechanism needs to be chosen. Most of the widely used machine learning algorithms operate on static, non-interactive data sets which have already been collected offline, building a model from one data set and then using that model to assess a new data set drawn from the same data source, predicting unknown values from known ones. This process describes the a supervised machine learning problem. However, in the case of optimizing runtime interoperability, these datasets are not available for training, making a pure supervised method unusable. Rather, the agent is supposed to learn online based on the experience he makes by acting inside the IoT environment. ML methods that fall into this area are collectively described as *reaction learning* where the agent has to adapt his learned knowledge constantly from the environmental stimuli to his actions(reactions).

One of the most prominent examples of reaction learning is *Reinforcement learning* (RL). Unlike other machine learning algorithms, reinforcement learning algorithms are capable of dealing with not only non-interactive data, but interactive processes in which classification decisions must

be made on the fly that can potentially affect which data will be gathered and classified in the future. The I-IOP agent is designed to be used with such an interactive learning procedure which receives IoT environment situations as input and needs to learn correct actions to optimize the interoperability process. Actions are rewarded based on the utility of the resulting state and the interoperability score, which is calculated through the interoperability metric as explained before. RL is a large field with considerable progress in the last couple of years. Generally RL methods can be split into value-based and policy-based methods with one of the most commonly used approaches being the Q-learning algorithm, a value-based RL method. Q-learning learns a Q-table which contains values for each state-action pair which indicate the accumulated future reward if the agent takes a particular action in a particular state. The Q-values are updated constantly as the agent explores the environment which will ultimately lead to convergence of the optimal Q-function under certain assumptions [TMS17].

A related class of reaction learning approaches are Learning classifier system (LCS). Learning classifier system (LCS) combine elements of supervised and reinforcement learning and evolutionary algorithms in order to derive rules, how to act in an environment. Advantages of such rule-based machine learning systems are (according to [Ste17]):

- The decision process and the trained policy is comprehensible for humans
- Actions are explicitly assigned to certain situations
- Exploration is restrictable by introducing similarity measures on the situation descriptions
- A fitness value is explicitly assignable to actions with a direct situational context

The objective of a LCS algorithm is to optimize payoff based on exposure to stimuli from a problem-specific environment. This is achieved by managing the credit assignment for those rules that prove useful and searching for new rules and new variations on existing rules using an evolutionary process [Bul15]. The general idea behind LCS is, that in complex systems it is easier to model the system as a set of rules than having one best fitting model [UM09]. Since their introduction, a large number of LCS algorithms have been developed (for a recent survey, the reader is referred to [UM09]). One can generally distinguish between the *Pittsburgh-style* and *Michigan-style* classifier systems [WvO12]. The difference between the two lies in the representation of individuals in the classifier population, where Pittsburgh-style evolves rule sets while in Michigan-style approaches one rule set (population of classifiers) is learned [But15]. One of the

most popular Michigan-style classifiers is the *XCS* algorithm [WvO12] (see figure 6-8) which is used in the I-IOP agent implementation since it strives to evolve maximal general problem classifiers that are maximally accurate [But15]. The XCS classifier system is shown in figure 6-8. It contains a population of classifiers which applies to one or multiple states of the environment (condition). A simple representation for such conditions is obtained by using binary state features 0,1 and #, where '#' is considered a wildcard. For example, a classifier with condition '01#00' maps to states '01000' and 01100. Each classifier contains actions which specify what action the agent applies when a classifier is matched. The population is matched during the encounter of a new problem instance. If actions are missing from the match set, they are added through a so called *covering* mechanism [But15]. XCS then estimates the payoffs for the actions in the match set and chooses the most promising action which is subsequently executed in the environment. An action set is formed containing all classifiers that contain the chosen action. After feedback from the environment is received, it is used to update the action set parameters and the process is continued for the subsequent iterations. The population, match set and action set are constantly modified through a steady state genetic algorithm to explore the problem space [Ste17].

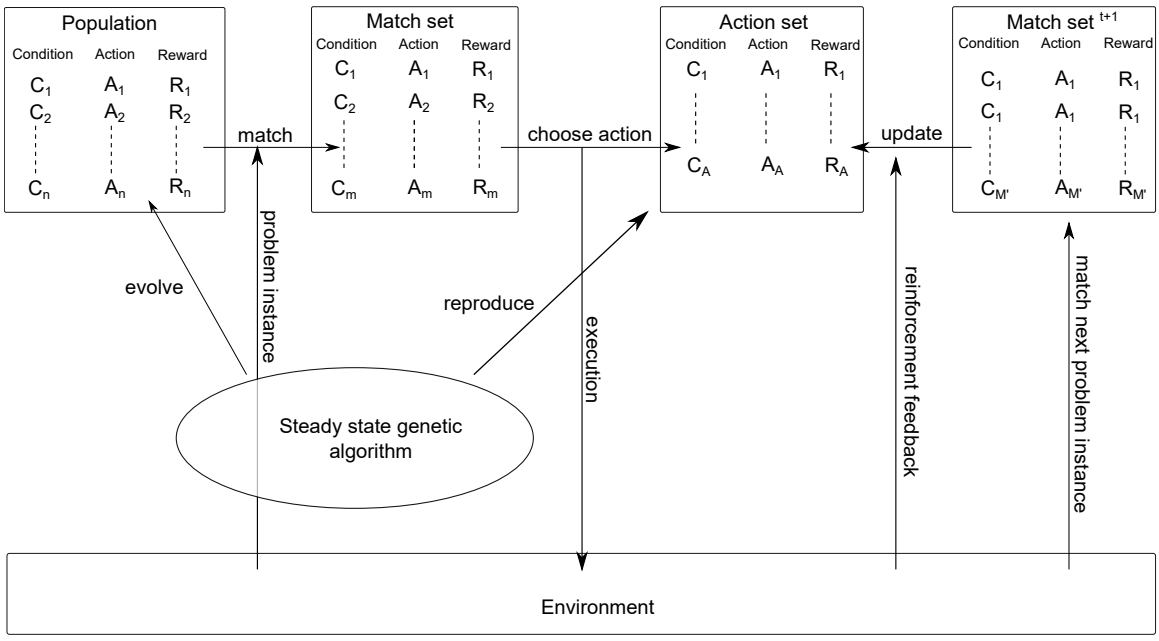


Figure 6-8: The process of the XCS online classifier system as defined in [But15] - pp.966

The following attributes make XCS a preferred choice for the learning module of the I-IOP agent [TMS17]:

- It is uniquely suited for dynamic environments
- It is adaptive - it accommodates to changing environments and changes in problem space
- It is model-free - it limits assumptions about environment (environment refers to source of training instance)
- It learns in clean or very noisy problems
- It accommodates missing training data
- The training results are interpretable

The integration of the learning module into the I-IOP architecture requires the agent to perceive environment stimuli, which is possible through the monitoring component. The environment situations, actions and perceived feedback is processed by the learning module of the controller component. Feedback is calculated through the interoperability metric, which defines how *well* a particular state of interoperation fits the IoT agent's utility. The agent applies the presented XCS based learning mechanism in an online fashion to update its underlying policy, i.e. rule-base in order to improve the interoperability. In this way, each I-IOP agent learns to act in the given environment according to the developer/customer intentions. Since the policy and learning module are responsible for adaptive behavior, they achieve properties SADP-2 and SADP-3.

6.3.4 Architectural integration

The integration of the I-IOP agent with existing systems is the crucial aspect to introduce runtime interoperability. Obviously, in order for the I-IOP architecture to work at its maximum level, it needs to be properly integrated with existing IoT systems. This requires from the agent middleware to be easily integrable in the first place. If integration is too cumbersome, IoT developers will not adopt the solution. The practical integration will be explained in this section in the context of the BIG-IoT project.

BIG IoT Project background The goal of the the BIG-IoT project [3] was to bridge the interoperability gap between IoT platforms. Multiple industrial and academic partners gathered as part of the European IoT Platforms Initiative to create an interoperability lifecycle instance, consisting of a marketplace and a SDK for semantic interoperability between IoT platforms. The domain for the BIG IoT project was settled in the Smart City context where the heterogeneity of IoT platform offerings becomes especially apparent.

The purpose of the BIG IoT marketplace is for matchmaking between requests and offers and thus fulfills the role of a central Infrastructure provider (INSP) agent. It provides a service discovery mechanism and allows providers to register offerings on the marketplace, which contains (among other components) a service registry. The offerings are described semantically by the providers and describe various types of digital IoT services such as: Parking space data / parking reservations services, environmental data or charging station data. The semantic descriptions are based on common and custom designed ontologies from the IoT sector. By extending existing ontologies, it was made sure that proper interoperability with existing systems is possible. It also allows to register higher-level services which provide, for example routing functionality. These services can be composed through recipes to form higher order services. The semantic annotation meta-data, called semantic descriptions, allow to find the offerings by the DSC agents using a semantic query engine (SPARQL).

The transaction functionality of the interoperability module is implemented through an API which is delivered as a Java-SDK for easy distribution and integration into existing IoT platforms. The API depicts the main functionalities needed for communication between IoT agents. Specifically, the following functions are provided by the API:

- Authentication (M1) - for authenticating with the BIG IoT marketplace (used by DSC & DSP agents)
- Registration (M2) - for registering offerings on the BIG IoT marketplace (used by DSP agents)
- Discovery (M3) - for discovering offerings on the BIG IoT marketplace (used by DSC agents)
- Subscription (M4) - for subscribing to an offering of DSP (used by DSC agents)

[3] http://big-iot.eu/, accessed 29.11.2019

- Accounting (M5) - for accounting based on used offerings (used by DSC & DSP agents)
- Access (A1) - for executing the transaction functionality to retrieve data from an offering (used by DSC agents)

Furthermore, the project developed different versions of the SDK, to be used by different types of IoT platforms, from cloud-level to device-level platforms.

BIG IoT - I-IOP agent integration Figure 6-9, presents the integration of the I-IOP agent with the BIG IoT architecture. The BIG IoT SDK integrates with the BIG IoT marketplace through the M1-5 interfaces. The BIG IoT compliant IoT platforms integrate the BIG IoT SDK via the A1 interface. The I-IOP agent architecture acts as a middleware between the IoT infrastructure and the IoT applications. This allows different IoT platform services and applications to be jointly connected through the agent-based service-oriented middleware. The middleware provides all the essential requirements to enable runtime interoperability. In this way, digital services deployed on different IoT platforms are made interoperable through the implementation of the interoperability module. The interoperability module can also be implemented by a single INSP agent in a broker model (as explained previously). Due to the already mentioned deficiencies of this approach, it will not be considered hereafter. The same applies to the AC module which can be integrated into the individual DSC and DSP agents or into a central INSP agent. In the former case, the AC module will operate as explained previously and adapt the operation of the transaction module. The AC-module is connected to the transaction module through the I1 interface. In this way, transactions that are exchanged through the BIG IoT SDK can be monitored and controlled by the AC module of the I-IOP agent.

To better illustrate these transactions, an interaction protocol is provided in figure 6-10 which displays the essential steps during the information exchange between DSC, BIG-IoT marketplace and DSP agents.

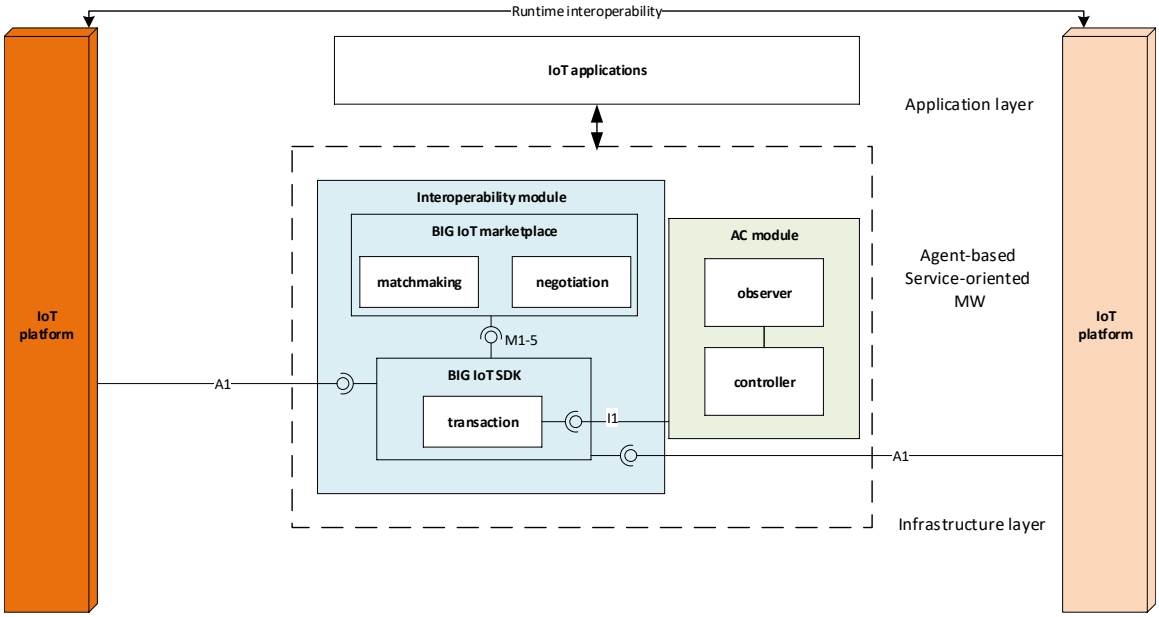


Figure 6-9: The integration between the I-IOP architecture and the BIG IoT project. The interoperability module consists of the BIG IoT building blocks with the AC module connected to the BIG IoT SDK via the I1 interface.

First, the DSC agent's interoperability module sends a discovery request to the BIG IoT marketplace, while the DSP agent commits a register request to register a digital service. The DSC agent receives a list of service offers which are fowarded, for selection to the system logic. The logic selects the relevant service(s) for operation which triggers engagement in the negotiation protocol between the DSC agent and the DSP agent. When the negotiation is finished, both agents receive a contract which defines the terms and conditions of their interaction, in particular the price, used ontology and protocol. The contract is forwarded to the AC module of the respective I-IOP agents. When, the DSC agent receives a request from the system logic, it sends a semantically embedded request (embedded in the negotiated ontology) to the DSP agent which uses its semantic and pragmatic processes to create a response to return back to the DSC agent. The response is processed by the semantic and pragmatic processes of the DSC system logic and the resulting state change is registered by the I-IOP agents and processed through the AC module. The AC module determines the necessary adaptive actions which are sent back to the interoperability module. The interoperability module then performs the action which will result in a new interoperability state for the subsequent interaction.

To make this integration work in practice, it needs to be assumed that the developer of an IoT system introduces the following elements in order for the I-IOP agent to work properly:

- A utility function which allows to relate situations to actions, needed for the learning module
- Implementation of adaptation logic
- Specification of an action set for the agent to execute
- The ability to embed communicated content into existing ontologies

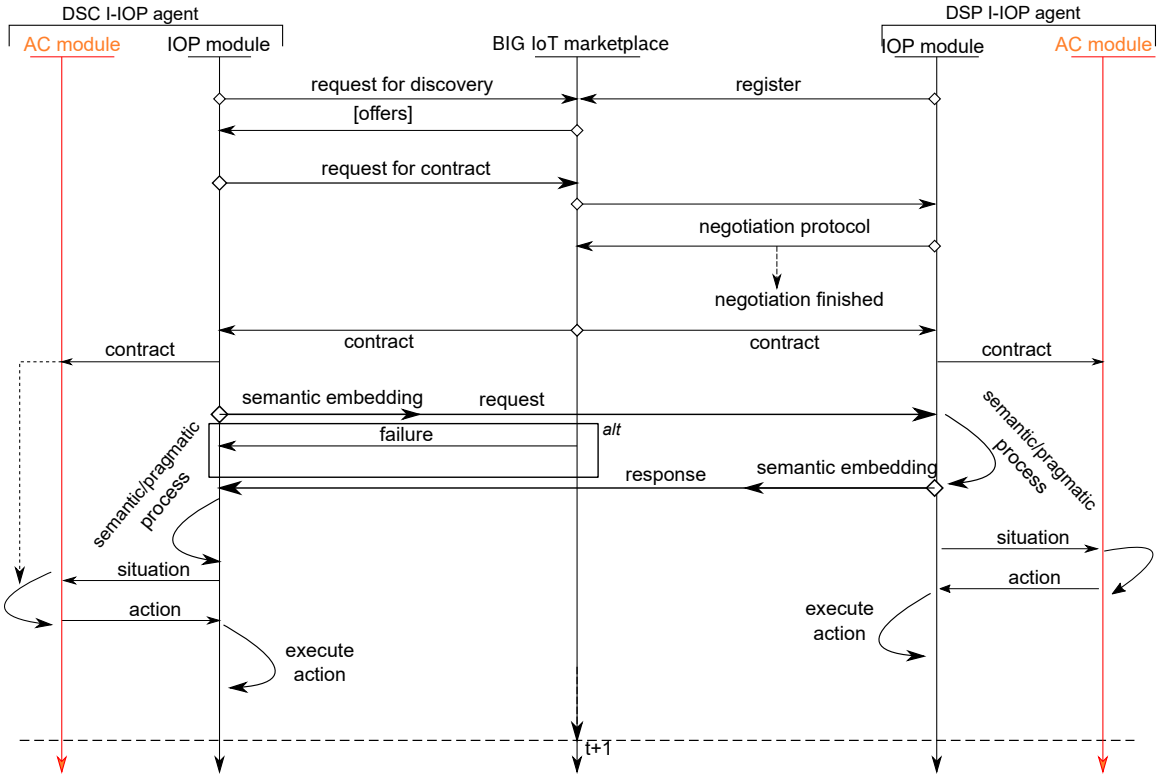


Figure 6-10: Interaction diagram showing the BIG IoT lifecycle implementation. The lifecycle has been adopted based on the BIG IoT project architecture.

6.4 Summary

This section has picked up on the theory of runtime interoperability from the previous chapter and gave a practical guide for an architectural model of an I-IOP agent to solve the runtime interoperability problem. In particular, different architectures for the implementation of an interoperability lifecycle were presented and compared. An architecture for the AC module was given as well that facilitates IoT agents to autonomously optimize the transaction optimization problem. To verify the architecture, the following chapter will describe an evaluation of the I-IOP architecture in the context of different IoT ecosystem scenarios.

Part III

Evaluation

Chapter 7

Empirical evaluation of the I-IOP agent architecture

7.1 Aim

This section empirically evaluates an implementation of the I-IOP agent architecture against the runtime interoperability requirements inside a simulated IoT ecosystem. The main question to answer in this context is, if the developed IoT agent model from the previous section implements all requirements and therefore presents a valid contribution towards solving the interoperability problem between IoT platforms. The general underlying hypothesis states that through the application of the I-IOP agent middleware, runtime interoperability as introduced in chapter 5 can be established. This hypothesis is evaluated through a feasibility study in a multi-agent based testbed which is able to simulate different IoT ecosystem compositions and scenarios and measures the necessary metrics to evaluate the interoperability results quantitatively. Pursuing a simulated approach has the following benefits, compared to an implementation within a *real* systems:

- Easier abstraction for the purpose of simplification
- Testing of hypotheses when real world testing not possible due to constraints and availability of data sources
- Adapt parameters to the specific problem

Developing a new simulator instead of choosing an existing one is necessary as there is no general available all-purpose simulator that can be used to create a detailed representation of an

end-to-end IoT service composition [CBBZ18]. Since the focus of this thesis is on the analysis of interoperability, the simulation was developed with a focus on this particular matter to show specifically, how the parameters of the IoT ecosystem affect interoperability.
The first section will describe the approach for the feasibility study. The proceeding section 7.2.1 introduces the problem setting for the simulator alongside measurable metrics which are used for later analysis. Section 7.2.2 describes the simulation model followed by section 7.2.3 which goes into the details of the implementation and the settings configurations.

7.1.1 Approach

The simulation approach for the feasibility study can be distinguished into three phases [LK91].

1. A **problem definition** is required which clearly states the problem to be analyzed alongside the **metrics** which are used for evaluation.

2. The **simulation model** is designed in an abstract fashion. The simulation model will be based on the IoT ecosystem concept from chapter 4

3. **Implementation** of the simulation model in some kind of software artifact and definition of inputs and parameters for running the simulation.

7.2 Feasibility study design

Since the IoT is a complex domain and for this study an ecosystem with multiple consumers and providers is assumed, such a system can easily become quite complex and difficult to analyze due to the various side effects. The here presented simulator thus has the purpose to make it possible to abstract all use case specific properties of real-world IoT systems to focus on the interoperability problem and specifically on the essential barriers for runtime interoperability. Since the simulator is **not** supposed to solve a use case specific problem, no domain specific information should be influencing the simulation. Rather, the goal is to show how the agents react and interact in a controlled environment without side effects to different types of interoperability problems. This rules out a physical, "real-world" testbed due to exactly these unwanted side effects. A further benefit of complexity reduction is that the results become less prone to outliers and are easier to reproduce.

7.2.1 Problem definition

For the problem definition, the use case studies from chapter 4, section 4.3.1 are re-examined. Although the purpose of this evaluation is not to solve a specific case study problem with an operational software product, the use cases are well suited to demonstrate different forms of IoT ecosystem. The case studies facilitate the definition of different scenarios that are supposed to demonstrate a set of typical interoperability patterns which usually occur in IoT ecosystems. The scenarios will then demonstrate if the IoT agent is able to deliver on the runtime interoperability requirements and thus solve the interoperability problem in IoT ecosystems. For some of the scenarios multiple agent configurations will be tested that change the constitution as well as the operation mode of the agent (e.g. by changing its learning module). More details on the parameters subject to change can be found in the next section. In particular, the smart industry (section 4.3.1.1) and smart agriculture (section 4.3.1.2) interoperability problems are analyzed in the study. To reiterate, these are:

- Agriculture - Smart farming: The interoperability goal is for the farming machines and equipment from different providers to work seamlessly together with the rest of the farming equipment. The manager of a Farm Management Information System (FMIS) needs to send and retrieve real-time data from his machines. The interoperability problem is thus a dual problem of actuation and sensing of IoT data.

- Industry - Smart production: The interoperability problem lies in the correct and real-time provisioning of process parameters to machines, since data has to be made available on demand quickly as well as custom-fit.

7.2.1.1 Scenarios

Each individual simulated scenario is chosen to highlight one specific interoperability-related situation inside the ecosystem scenario. Figure 7-1 shows these situations in an overview figure.

The scenarios originate from the proof-of-concept results of the BIG IoT project (see section 6.3.4). Since in the BIG IoT project real world use cases have been implemented, tested and evaluated, this information can be used as a reliable source for interoperability related problems in the complex of multi-platform ecosystems. The analysis was performed manually by outlining, for each use case, the specific problem to be solved by the interoperation, the participating systems and the information flows between the interfaces of the participating systems. The results of this

analysis were aggregated to create a list of common interoperability patterns (figure 7-1). To apply the patterns also to other domains outside Smart city, they were generalized and modeled as scenarios to be used in the simulator to test the I-IOP agent architecture.

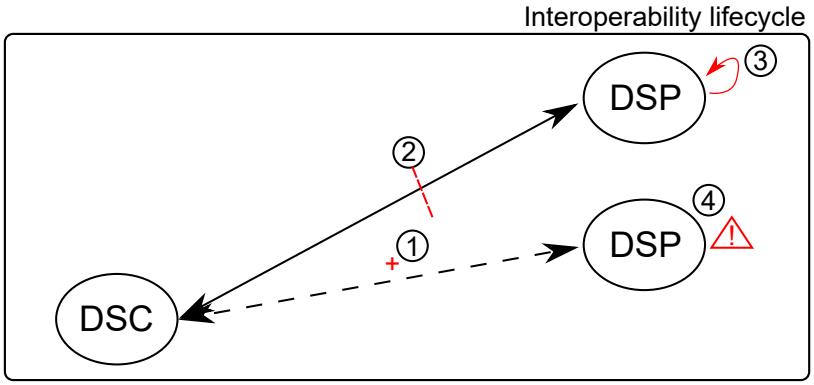


Figure 7-1: Interoperability related situations which can occur inside an IoT ecosystem in an interoperability lifecycle instantiation.

Figure 7-1 presents all scenarios, in order from (1) to (4) they are: (1) Addition of an interoperation partner, (2) interoperability recovery, (3) interoperability recovery with adaptation, and (4) failures of interoperation partners. More specifically, the exact procedure of each scenario is detailed in figure 7-2 and figure 7-3. Through the simulation of these patterns, the I-IOP agent's operation can be evaluated including the bounds of the architecture, i.e. at what point the agent can not achieve a further interoperability improvement. The success of the agent's performance with respect to the runtime interoperability requirements will always be quantitatively measured through the semantic and pragmatic interoperability metrics. The subsequent tables will list in detail the exact scenarios that have been simulated, including the participating systems and information flows.

Scenario description		
Identifier	S_1	Autonomous discovery
Summary	A DSC system needs to contract two DSP systems to operate.	
Scenario sequence	**Step**	
	1	DSC and DSP-1 are created & DSP-1 registers offering.
	2	DSC queries for providers and is matched with DSP-1.
	3	DSC and DSP-1 negotiate interoperability contract.
	4	DSP-2 is added after 5 iterations.
	5	DSC queries for missing providers and is matched with DSP-2.
	6	DSC achieves maximum level of interoperability by negotiating interoperability contract with DSP-2.
Interoperability evaluation	The scenario will show that how, through the interoperability lifecycle, the second DSP system is discovered & integrated to achieve the desired level of interoperability	
Runtime IOP requirements	autonomy, discoverability, service orientation	

Scenario description		
Identifier	S_2	Offline recovery
Summary	A DSC interoperates with two DSP systems, while one of them becomes offline and needs to be replaced with another DSP system.	
Scenario sequence	**Step**	
	1	DSC, DSP-1 and DSP-2 are created & DSP-1,2 register offerings.
	2	DSC queries for providers and is matched with DSP-1 and DSP-2.
	3	DSC negotiates interoperability contract with DSP-1 & DSP-2.
	4	DSP-2 is removed after 5 iterations.
	5	DSP-3 registers its offering.
	6	DSC queries for missing providers and is matched with DSP-3.
	7	DSC achieves maximum level of interoperability again by negotiating interoperability contract with DSP-3.
Interoperability evaluation	The scenario will show if in the case of DSP-2 becoming offline, DSC will autonomously achieve the desired level of interoperability again through the continuous querying for DSP systems.	
Runtime IOP requirements	autonomy, service orientation & discoverability	

Scenario description		
Identifier	S_3	Interoperability recovery with transaction adaptation
Summary	A DSC interoperates with two DSP systems, while one of them becomes offline and needs to be replaced with another DSP system that has to be adapted to be interoperable with the DSC.	
Scenario sequence	**Step**	
	1	DSC, DSP-1 and DSP-2 are created & DSP-1,2 register offerings.
	2	DSC queries for providers and is matched with DSP-1 and DSP-2.
	3	DSC negotiates interoperability contract with DSP-1 & DSP-2.
	4	DSP-2 is removed after 5 iterations.
	5	DSP-3 registers its offering.
	6	DSC queries for missing providers and is matched with DSP-3.
	7	DSC and DSP-3 adapt transaction channel.
	8	DSC achieves maximum level of interoperability again by negotiating interoperability contract with DSP-3.
Interoperability evaluation	The scenario will show if in the case of DSP-2 becoming offline, DSC will autonomously achieve the desired level of interoperability again by querying for replacement DSP systems and subsequent transaction adaptation.	
Runtime IOP requirements	autonomy, adaptivity, service orientation & discoverability, interoperability reasoning & measurability, semantic interoperability	

Figure 7-2: Description of the interoperability scenarios 1 to 3 which are evaluated in the feasibility study.

Scenario description		
Identifier	S_4	Perturbation resistance
Summary	A DSC interacts with a DSP which experiences random perturbations at runtime.	
Scenario sequence	**Step**	
	1	DSC and DSP-1 are created & DSP-1 registers offering.
	2	DSC queries for providers and is matched with DSP-1.
	3	DSC negotiates interoperability contract with DSP-1.
	4	DSP-1 experiences random pertruberations, (i) random unavailability, (ii) firmware updates, (iii) communication delays, and (iv) random communication failures
	5	DSC maintains interoperability with DSP through adaptation of transaction process.
Interoperability evaluation	The scenario shows if DSC is able to register interoperability related perturbations and is able to reach the desired level of interoperability again.	
Runtime IOP requirements	autonomy, adaptivity, service orientation & discoverability, interoperability reasoning & measurability	

Scenario description		
Identifier	S_F	Smart farming - high heterogenity
Summary	A DSC integrates a list of DSP systems which provide smart farming related services. Due to the large heterogenity of the systems, semantic and pragmatic adaptations are necessary.	
Scenario sequence	**Step**	
	1	DSC, DSP-1, DSP-2 and DSP-3 are created & DSP-1,DSP-2,DSP-3 register their offerings
	2	DSC queries for providers and is matched with DSP-1, DSP-2 & DSP-3
	3	DSC negotiates interoperability contract with all three providers
	4	DSC adapts its semantic interoperability mechanism towards DSP-2. DSP-2 also adapts its semantic communication mechanism.
	5	DSC adapts its pragmatic interoperability mechanism towards DSP-3. DSP-3 also adapts its pragmatic communication mechanism.
Interoperability evaluation	The scenario shows if the DSC-agent is able to achieve semantic and pragmatic interoperability in an exemplary instantiation of a Smart farming IoT ecosystems.	
Runtime IOP requirements	autonomy, adaptivity, service orientation & discoverability, interoperability reasoning & measurability, semantic & pragmatic interoperability	

Scenario description		
Identifier	S_P	Smart production - high precision
Summary	A DSC interoperates with two DSP systems in a smart factory. Both of the DSP systems are configured to provide maximum accuracy during the interoperation.	
Scenario sequence	**Step**	
	1	DSC, DSP-1 & DSP-2 are created & DSP-1,DSP-2 registers offerings.
	2	DSC queries for providers and is matched with DSP-1,2.
	3	DSC negotiates interoperability contract with DSP-1 & DSP-2.
	4	DSP-1,2 adapts its pragmatic interoperability mechanism to provide maximum accuracy with regards to the utility of the DSC system.
Interoperability evaluation	The scenario shows that the IoT-OC agent of the DSP systems is able to adapt interoperation so that the utility of the DSC systeme is maximized. This scenario thus specificall addresses pragmatic interoperability.	
Runtime IOP requirements	autonomy, adaptivity, service orientation & discoverability, interoperability reasoning & measurability, pragmatic interoperability	

Figure 7-3: Description of the interoperability scenarios 4 to S_P which are evaluated in the feasibility study.

7.2.1.2 Metrics

In order to quantify the results and to judge the performance of the I-IOP agent in optimizing the transaction phase of the interoperability lifecycle, the semantic and pragmatic interoperability metric are used, which have been introduced in section 5.6. The reasoning being that semantic and pragmatic interoperability are the key components of the reward measure for I-IOP agents in runtime interoperability which was defined in definition 5.32 in section 5.6. Thus, if a minimization of the interoperability gap between IoT agents can be quantified through applying the semantic and pragmatic interoperability metrics, the performance of the agents with regards to establishing runtime interoperability can be derived - which is ultimately the goal to evaluate the soundness of the I-IOP agent architecture. To apply these metrics in the simulator, a couple of measurement points have been selected to derive the relevant state information from each transaction between two agents to measure, for example, the desired state vs. the actual state for the pragmatic interoperability metric or the occurrence of ontology mismatches when agents exchange semantic information in the simulated environment.

Cost of self-awareness The application of an I-IOP agent architecture always comes at a cost of performance due to the increased overhead compared to running the same system without a dedicated agent. Thus, to assess the performance and the cost of the agent architecture, the following performance parameters are additionally measured during the simulation:

- The memory allocation for each I-IOP agent in MB
- The communication overhead of each message exchange for SUOC-to-agent communication and agent-to-agent communication (in number of messages and message size in MB)
- The execution time of the agent logic and of the training process for the XCS learning module (in seconds)
- The number of times the agent executes an action on the underlying SUOC (in number of actions)

7.2.2 Simulation model

Each scenario is run for a pre-determined number of iterations, which differs based on the scenario being analyzed between 10-20 iterations. The simulation model implements the scenario specifi-

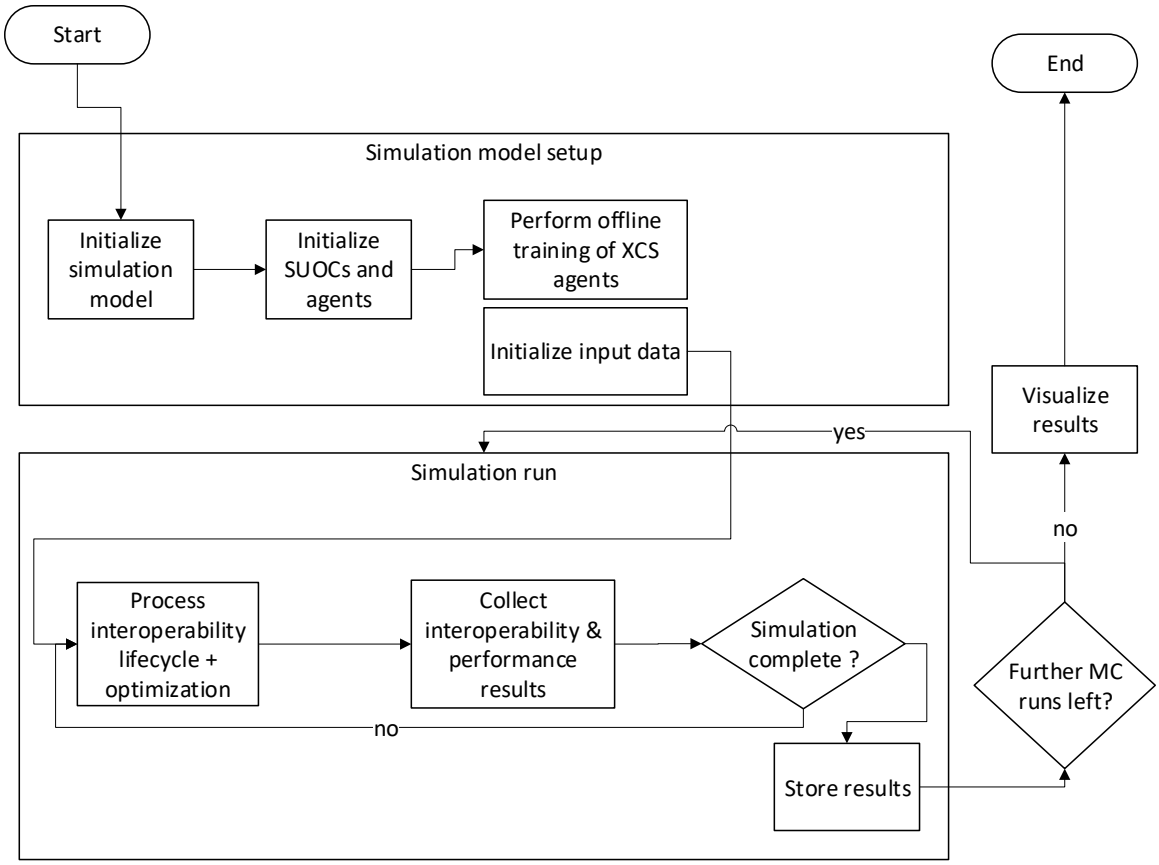


Figure 7-4: The activity diagram describing the simulation flow in the feasibility study. It is distinguished between the main blocks of simulation model setup and simulation run.

cation which means that it consists of an Infrastructure provider (INSP) agent, one or more DSC and DSP SUOC implementations and a varying number of I-IOP agent instances. Additionally a customer is simulated who has a concrete use case related problem which is manifested in a particular simulation space configuration. The customer input to the DSC agent is simulated through artificial *interactions* of the customer with the DSC agent.

The flowchart in figure 7-4 describes the flow of the simulation in more detail. It starts with the simulation model setup step which initiates the simulation model, the SUOC and agents and triggers the training phase for the XCS module. Also in this phase, the artificial customer input is generated which is the used to run the simulation ticks. During each simulation tick, the interoperability lifecycle is processed as well as any adaptations by the agents. Certain metrics, which are explained in the next paragraph, are measured and after that the next simulator iteration starts unless the maximum number of iterations is achieved. If this is the case, the results are stored and

visualized which ends the simulation.

7.2.2.1 Strategies

The simulator can be equipped with a number of pre-defined strategies that the I-IOP agents can use either for learning a policy or as an expert rule-base. The strategy thus defines the *trait* of the agent. The strategies determine, based on the observed situation what semantic and pragmatic action to execute. The standard strategy is the utility maximizing strategy which strives to always maximize the semantic and pragmatic interoperability in the simulator. Yet, the agents can also be initialized with another strategy, for example an *always-cost-minimizing* strategy or a strategy which always adapts to other agents.

7.2.2.2 Utilities

The simulator allows for different utility functions for each agent. The utility function is always split into two parts: A utility part and the metric part. The utility part defines a mask, which is overlain onto the received state description of the agent. The masked situation description is used inside the policy module to determine the most efficient action. The metric part is used by the interoperability state analyzer to create the situation description. It defines the thresholds that are needed for the binarization of the state space features. It is also important to highlight that of course all simulated agents have an underlying utility.

7.2.2.3 Simulation configuration

The simulation features a large configuration space for simulating a broad range of IoT ecosystem situations. Figure 7-5 highlights the configuration space for the simulation:

The environment configuration defines the interoperability lifecycle implementation, the type of agent deployment and the simulated customer instantiation. (Note that technically, the SUOC is also part of the environment but it was decided to separate it to make the configuration space discussion more accessible). The SUOC configuration space can furthermore be broken down into the parameters in table 7.1.

Random parameters are used to simulate the natural variance in an IoT ecosystem which is due to different types of consumer/provider or customer events. The agent configuration dimension determines, which of the simulated SUOC have an agent component attached, which learning

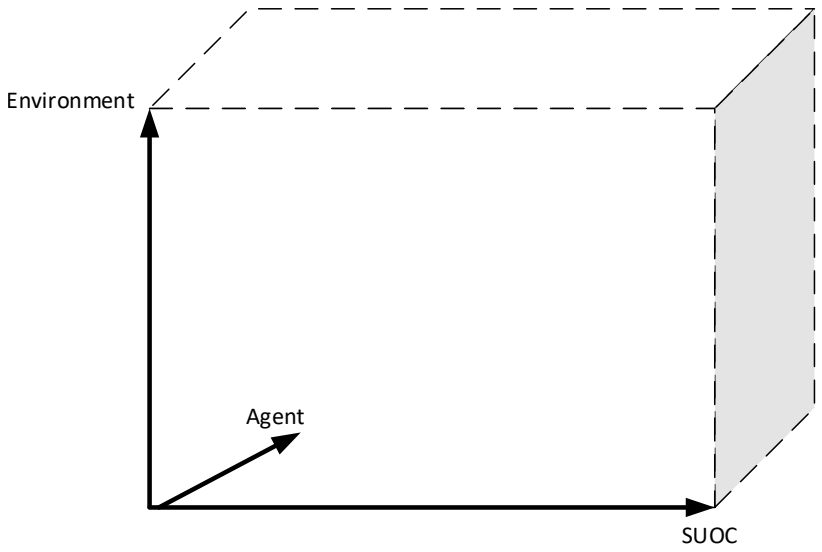


Figure 7-5: Configuration space of the simulation is partitioned into the environment, agent and SUOC axis.

DSP configuration	**DSC configuration**
API configuration	Request configuration
Ontology used	Ontology used
Provider utility function	Consumer utility function
Provider events	Consumer events
Number of providers simulated	
Degree of provider functional heterogeneity	

Table 7.1: Properties of the SUOC which are configurable in the simulator.

mode these agents use, what the communication costs are or if communication is turned off and what XCS configuration they use. An example XCS configuration is shown in table A.12.

SUOC adaptation parameters To analyze the changes in interoperability as the simulation progresses, it is necessary to simulate typical adaptations of SUOC. Based on the identified *typical* interoperability patterns in IoT ecosystems (7-1) a list of parameters are defined to be used by the AC module's controller and considered part of the simulation model specification. The following list contains the parameters of the SUOC to allow adaptive actions.

- Provider API configuration
  - Response type
  - Employed ontology
  - Employed protocol (e.g. request-response/ feed)
  - Response precision
  - Response rate
- Consumer configuration
  - Request mode
  - Request frequency
  - Employed ontology
  - Employed protocol

7.2.3 IoT ecosystem simulation implementation

The class diagram for the simulator implementation is displayed in figure 7-6. To keep the diagram readable, function arguments and return types are omitted. The simulator was implemented as a custom software artifact (due to the already mentioned unavailability of desired multi-agent simulation frameworks) in Python 3.6 and was run on an Intel core i7-6820HQ system with 4 X64-cores at 2.7 Ghz, 32 GB RAM on Windows 10 enterprise, build 17763. The implementation supports parallelization of the IoT processes through the Python built-in *multiprocessing* module.

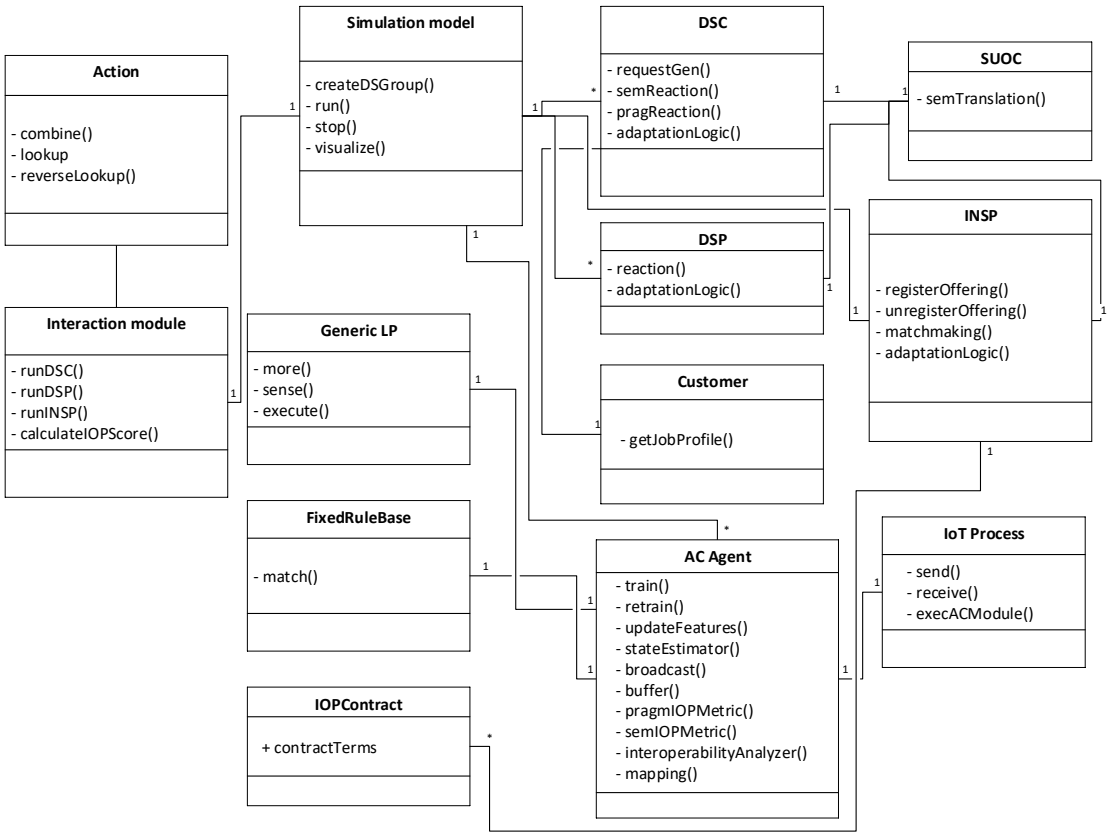


Figure 7-6: The UML diagram of the IoT ecosystem simulator.

7.2.3.1 BIG IoT lifecycle implementation

The simulator implements the BIG IoT architecture and I-IOP agent integration as described in section 6.3.4. The implementation approach follows a customer oriented perspective, i.e. the interaction is always triggered through the DSC agent. Since the purpose of this study is not to analyze the emerging structures through multiple DSC-DSP agent compositions, this approach is better aligned with the vision of the IoT to deliver optimal value for customers.

The implementation simulates the BIG IoT marketplace where, at the beginning of the simulation, the offers of DSP agents are registered and later matched against the DSC agent requests. However not all components of the *complete* BIG IoT marketplace are implemented - in this simulation only the core matchmaking functionality. Demand requests are created based on a simulated customer input, which has certain goals to be fulfilled by the interoperation. All DSC and DSP agents in the simulation are pre-configured with a direct communication connection to the mar-

ketplace. The matchmaking model follows a simplified model which compares the offering type and the number of offered input and output parameters with the requested offer. If all these elements match, the offer is added to the candidate list.
What is missing in the BIG IoT consideration is the automated contract negotiation, as part of the negotiation module. The simulation implementation thus adds an implementation of F_N in order to have a complete, automated interoperability lifecycle implementation for the IoT agents. The negotiation functionality negotiates a contract according to the customer request and the offerings through a simple version of the contract net protocol (see 6.2.2). The contract is stored by the INSP agent and sent to both the DSC and DSP agents who also store the contract on their end. During information exchange between the DSC and DSP agents, the semantic and pragmatic processes are executed by internal SUOC implementations. The implementations follow a simplified version of a real-world IoT system by abstracting the use case specific data. The transaction modules of the I-IOP agent simulate a service oriented communication channel through a message queue. Messages on this queue are interpreted as offering requests or responses by the I-IOP agents.
The interoperability score calculation is done after each simulated transaction between the IoT agents and stored for later visualization.

7.2.3.2 Learning module

The implementation of the learning module of the AC-module is based on the XCS implementation from [1]. The actual learning process is performed in the following way: The *Generic_LP* class defines the learning environment which is used by the training module of the I-IOP agent. The strategy class instance defines the pre-determined rule-base strategy for the agent and is used for training the XCS policy for a number of iterations. The trained policy is stored in the AC module for later use at runtime by the I-IOP agent.

7.2.4 Verification & Threats to validity

Although the simulation testbed implements the essential elements in an IoT ecosystem, the following disclaimers have to be kept in mind:

- Due to the real world abstraction, the testbed will not be completely accurate

[1] https://pypi.org/project/xcs/ , accessed 29.11.2019

- The outcome heavily depends on the customer input, thus it is essential to carefully design the input space so that it is representative for a real-world scenario
- All *events* and *threats* in the system, e.g. misbehaving or failing entities need to be implemented by the simulator

In order to make sure that the simulation model properly implements the conceptual model of an IoT ecosystem it needs to be verified that the implementation matches the specification of an IoT ecosystem and the I-IOP agent model. In essence, verification makes sure that the implementation of the simulation model is correct. For this purpose the implementation of the IoT ecosystem properties are discussed in table 7.2 to verify that the simulator implementation is in accordance with the defined IoT ecosystem properties in section 4.3.5 .

IoT ecosystem property	Verification
self-organization / openness	DSC agents are implemented in a way that they choose the contracts themselves based on their utilities / negotiation is implemented so that no pre-determined collaborations are assumed
loose coupling	The service descriptions/queries and interoperability contracts are used to implement the loose coupling between DSC and DSP agents. No direct couplings are pre-determined
domain-clustered	The simulation supports separate runs and configurations to simulate different domains (see scenarios S_F and S_P)
demand-driven	The simulation follows a customer-oriented focus. That means, the job profile of a DSC agent drives the whole interoperability process including the adaptation
belonging	The I-IOP agent process shows how the decision making of the IoT agent is taken to runtime instead of being pre-determined
connectivity	The Simulator allows to model events related to connectivity, e.g. a failing communication between IoT agents
emergence	*Emergence is not covered since the focus of this simulation is not on emergent behavior between a multitude of agents*

Table 7.2: Verification criteria that the IoT ecosystem properties are met by the IoT ecosystem simulator.

7.3 Results

This section reviews the results from the I-IOP agent evaluation. This part only covers the objective result analysis. The interpretation of results will follow in chapter 8. The semantic and pragmatic interoperability scores are visualized in an aggregated fashion using a Boxplot visualization and a detailed version using a line graph. This highlights, on the one hand, the overall measurable performance of each tested configuration and on the other hand the detailed runtime performance during the individual iterations. In this way, the imposed research question whether the I-IOP agent can solve the runtime interoperability problem can be answered. The result preparation process will be explained in detail for the first scenario, after that only the results will be presented since the process is the same for the rest of the scenarios. To ease readability, configurations are abbreviated using their configuration identifier (e.g. XCS $100k$,S_1) as a reference and can be found through the same identifier in the annex in section A.4.1. Also the terms *semantic interoperability* and *pragmatic interoperability* are shortened to SI and PI.

Different variants of agent configurations are used within the experiments: An agent setting with an expert rule system without an active XCS component, an active DSC and DSP XCS system with a 10.000 training cycles and 50.000 training cycles. Comparing these configurations will showcase the differences between distinct XCS deployments on the one hand and the performance of XCS compared to an expert rule system which functions as a benchmark. Further experiments were performed with different XCS configurations, showcasing the differences between varying the number of training cycles and the amount of exploration the agent is allowed to perform to cover the problem space. This will demonstrate the benefits of the GA component of the XCS module and in particular its ability to work in new situations experienced at runtime. During each simulation tick, the interoperability score is measured based on the current environment situation and stored by the simulator for later visualization. The values for the maximum SI and PI score were arbitrary chosen between six and zero.

7.3.1 Scenario S_1 results

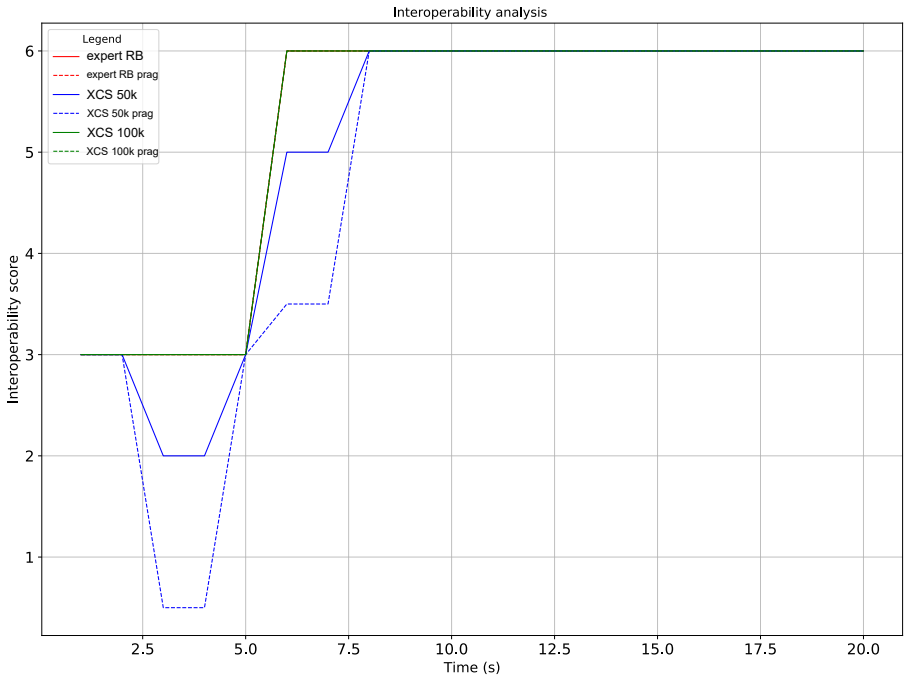
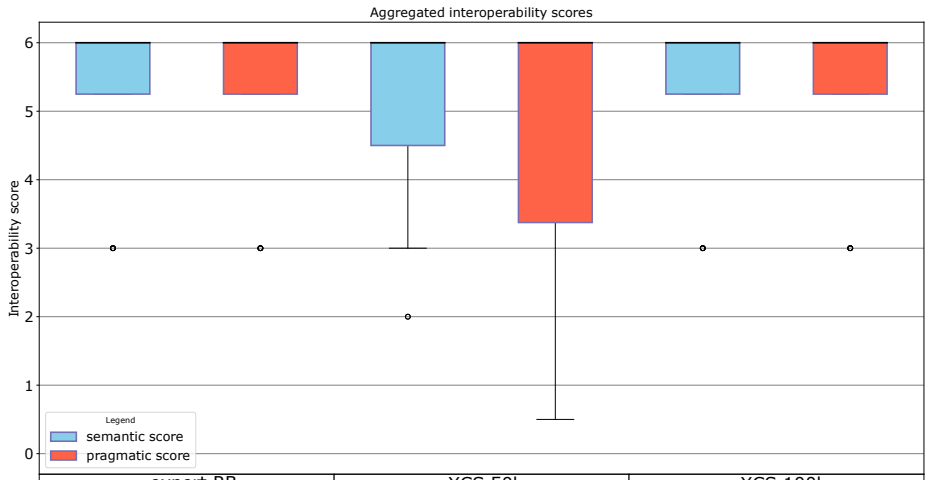


Figure 7-7: Interoperability analysis for scenario S_1 including SI (solid lines) and PI scores (dashed lines)



	expert RB	XCS 50k	XCS 100k
Median_semantic	6.0	6.0	6.0
Median_pragmatic	6.0	6.0	6.0
Mean_semantic	5.25	5.05	5.25
Mean_pragmatic	5.25	4.75	5.25
Std_semantic	1.3	1.47	1.3
Std_pragmatic	1.3	1.85	1.3

Figure 7-8: Aggregated representation of the SI and PI scores for scenario S_1. The table below each box presents the median,mean and standard deviation for each scenarios' SI and PI score.

The interoperability analysis in figure 7-7 lays out the SI (solid line) and PI scores (dashed line) on a timescale for each configuration. No normalization was performed, instead the values were kept at their original values for easier comparison. In the case of overlapping values between the graphs, only the results from the last executed configuration are visible. The horizontal axis of the graph refers to the iterations while the vertical axis shows the interoperability score on scale of zero to six.

Figure 7-7 shows an identical SI and PI score behavior for *expert RB* and XCS $100k$. For both configurations, an increase of SI and PI scores is noticed between iterations 5 and 6, which coincides with the activation of DSP-2 in the scenario. After iteration 6, the SI and PI scores for both configurations stay constant at the maximum interoperability level. The results for XCS $50k$ reveal an initial decrease in SI and PI scores between iterations 2 and 5 before gradually increasing to the maximum interoperability level at iteration 8.

Boxplot 7-8 summarizes the SI and PI scores, in particular the median, upper and lower quartile for each score alongside potential outliers for each configuration. The table below the plot contains the median, mean, and standard deviation of the interoperability scores for a more detailed analysis.

In the aggregated S_1 results, all configurations achieved a median SI and PI score of 6.0. *expert RB* and XCS $100k$ have identical values for the mean and standard deviation for SI and PI. XCS $50k$ achieved a slightly weaker mean SI and PI score at 5.05 and 4.75 and a slightly higher standard deviation.

Metric	Value
Execution time	9 m
Average message size (MB)	56
Number of sent messages between agents	213
Mean agent execution time (seconds)	0,001580645
Number of actions performed	47

Table 7.3: Experiment summary for scenario S_1

Table 7.3 lists the simulation experiment summary. The scenario took 9 minutes to complete with a total of 213 messages sent between the agents with an average message size of 56 megabytes and 47 actions performed. The mean agent execution time (i.e. the I-IOP agent internal logic) was measured at 0,001580645 seconds.

7.3.2 Scenario S_2 results

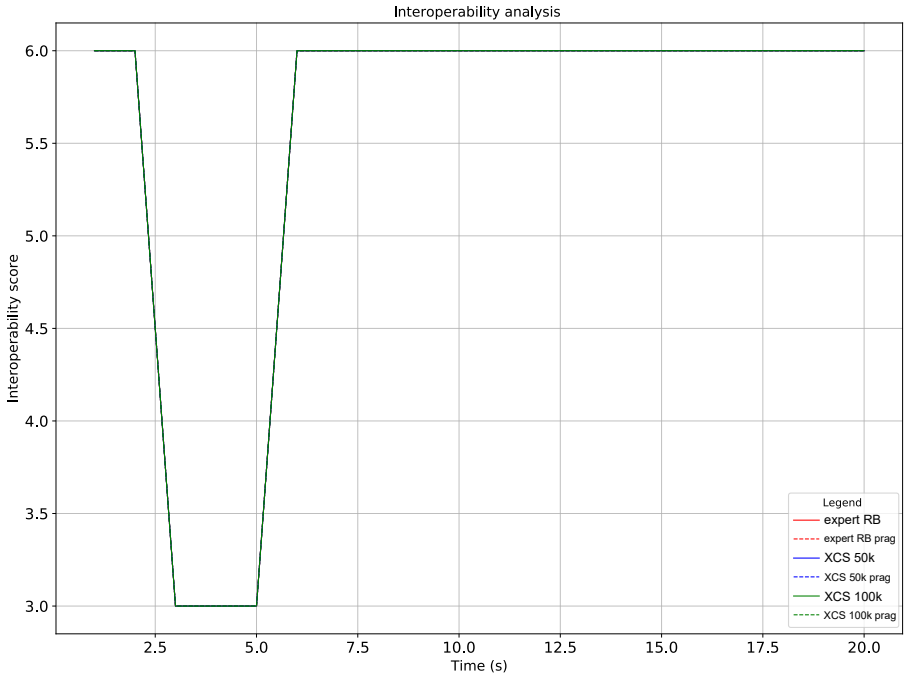
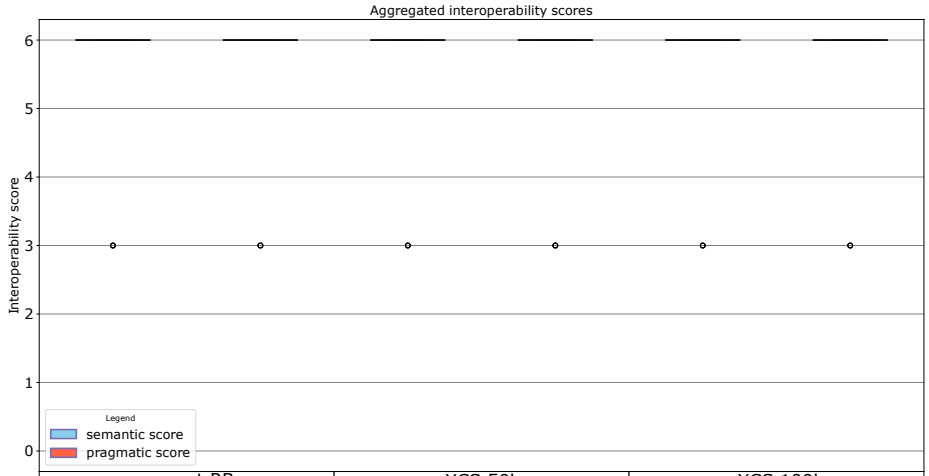


Figure 7-9: Interoperability analysis for scenario S_2 including SI (solid lines) and PI scores (dashed lines)



	expert RB	XCS 50k	XCS 100k
Median_semantic	6.0	6.0	6.0
Median_pragmatic	6.0	6.0	6.0
Mean_semantic	5.55	5.55	5.55
Mean_pragmatic	5.55	5.55	5.55
Std_semantic	1.07	1.07	1.07
Std_pragmatic	1.07	1.07	1.07

Figure 7-10: Aggregated representation of the SI and PI scores for scenario S_2

Metric	Value
Execution time	13 m
Average message size (MB)	56
Number of sent messages between agents	657
Mean agent execution time (seconds)	0,002348225
Number of actions performed	11

Table 7.4: Experiment summary for scenario S_2

For scenario S_2, the results in figure 7-9 and 7-10 depict an identical interoperability pattern for all three agent configurations. As seen in figure 7-9, the interoperability score drops to a medium level of 3 when DSP-2 operation is stopped. The score reaches the maximum level again after iteration 5 when DSP-3 is deployed and successfully matched with the DSC agent. The execution time for this scenario was measured at 13 minutes, with 657 messages sent between the agents with an average message size of 56 megabyte and in total 11 actions performed. The mean agent execution time was 0,002348225 seconds.

7.3.3 Scenario S_3 results

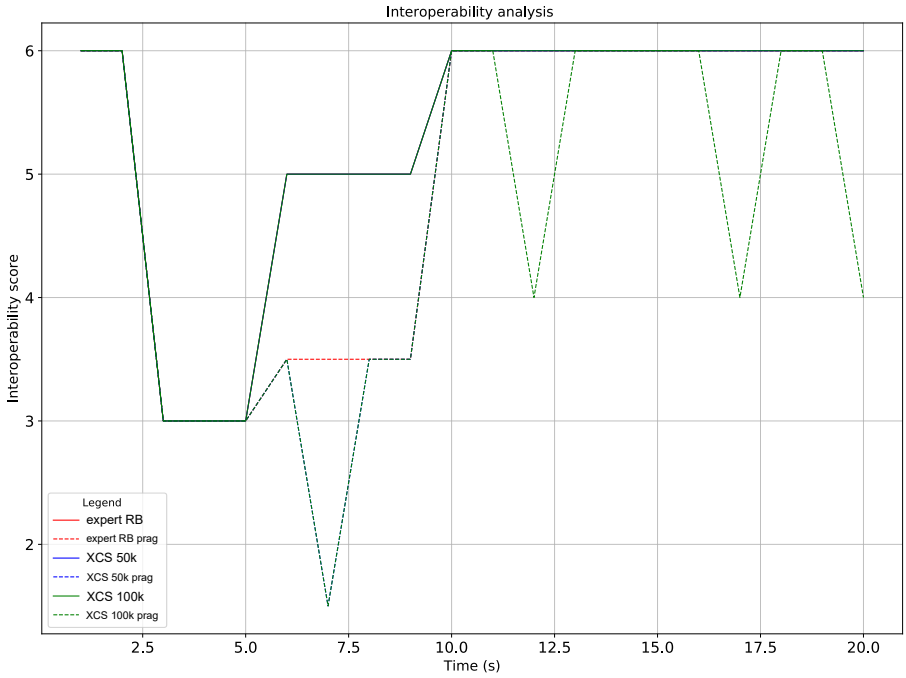
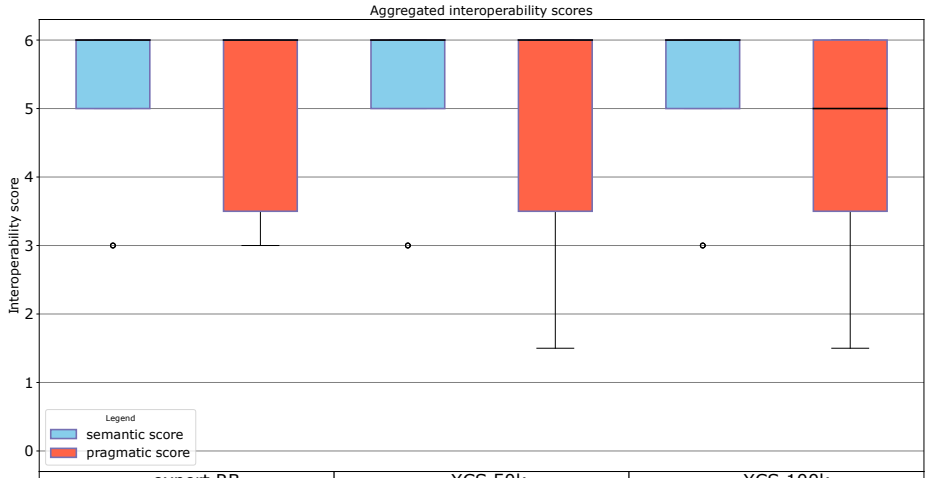


Figure 7-11: Interoperability analysis for scenario S_3 including SI (solid lines) and PI scores (dashed lines)



	expert RB	XCS 50k	XCS 100k
Median_semantic	6.0	6.0	6.0
Median_pragmatic	6.0	6.0	5.0
Mean_semantic	5.35	5.35	5.35
Mean_pragmatic	5.05	4.95	4.65
Std_semantic	1.06	1.06	1.06
Std_pragmatic	1.3	1.48	1.44

Figure 7-12: Aggregated representation of the SI and PI scores for scenario S_3

Metric	Value
Execution time	16 m
Used memory (MB)	56
Number of sent messages	655
Mean agent execution time (seconds)	0,115681024
Number of actions performed	78

Table 7.5: Experiment summary for scenario S_3

In scenario S_3 we notice an identical pattern of semantic interoperability between the three tested agent configurations. The interoperability scores decrease after iteration 2 and recovers after iteration 10. The PI scores for *expert RB* and XCS $50k$ achieve maximum values again at iteration 10, with a slightly lower score of 4 between iterations 6 and 10. The PI score of XCS $100k$ drops to the lowest score of 1 at iteration 8 before reaching the maximum score at iteration 11. Eventually, all configurations have reached maximum interoperability after iteration 11 but the PI score for XCS $100k$ oscillates repeatedly afterwards until the end of the simulation. Figure 7-12 reveals that *expert RB* and XCS $50k$ have achieved the overall same performance, while XCS $100k$ has a slightly lower mean PI score and a slightly higher standard deviation. The scenario execution took 16 minutes, 655 messages sent between agents (with a constant message size of 56 megabytes) and 78 actions performed in total. The mean agent execution time was measure at 0,115681024 seconds.

7.3.4 Scenario S_4 results

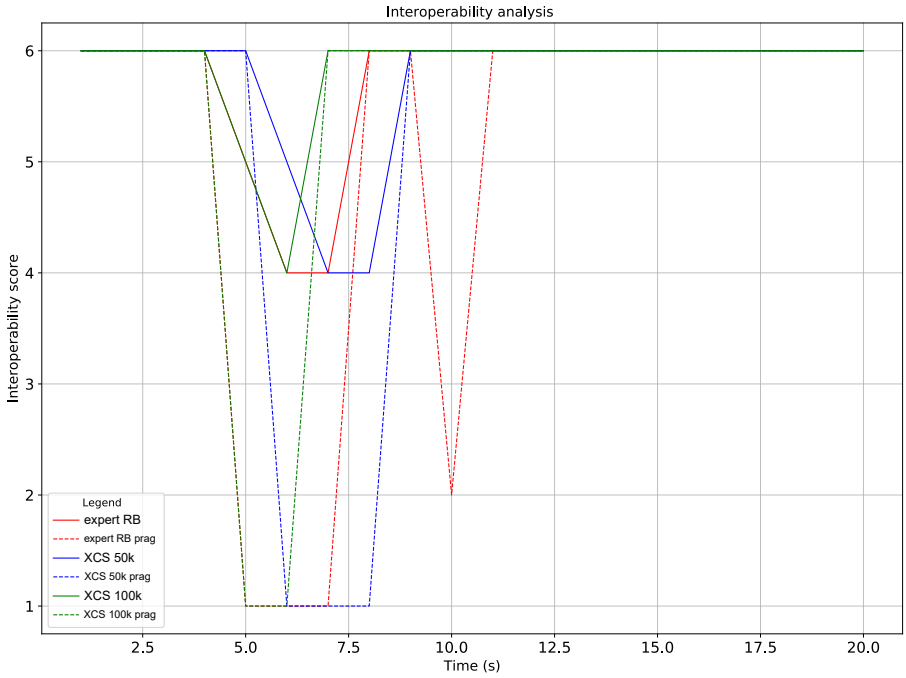
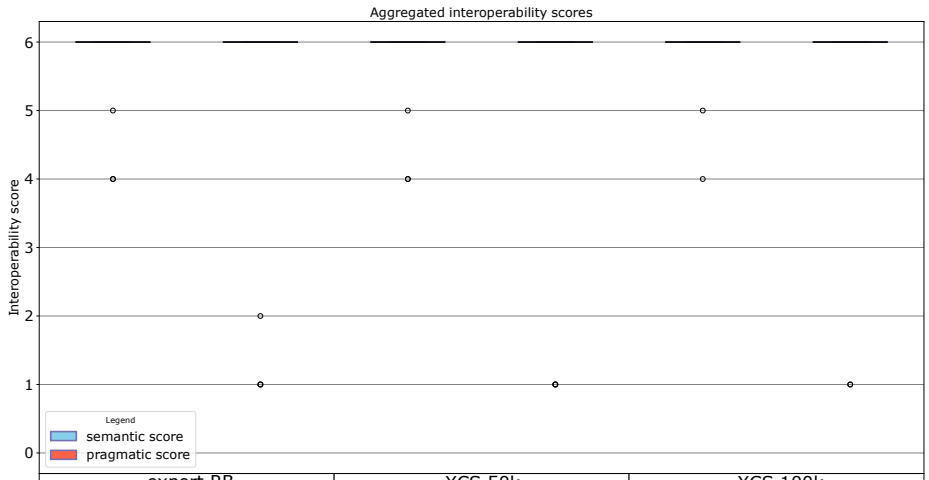


Figure 7-13: Interoperability analysis for scenario S_4 including SI (solid lines) and PI scores (dashed lines)



	expert RB	XCS 50k	XCS 100k
Median_semantic	6.0	6.0	6.0
Median_pragmatic	6.0	6.0	6.0
Mean_semantic	5.75	5.75	5.85
Mean_pragmatic	5.05	5.25	5.5
Std_semantic	0.62	0.62	0.48
Std_pragmatic	1.91	1.79	1.5

Figure 7-14: Aggregated representation of the SI and PI scores for scenario S_4.

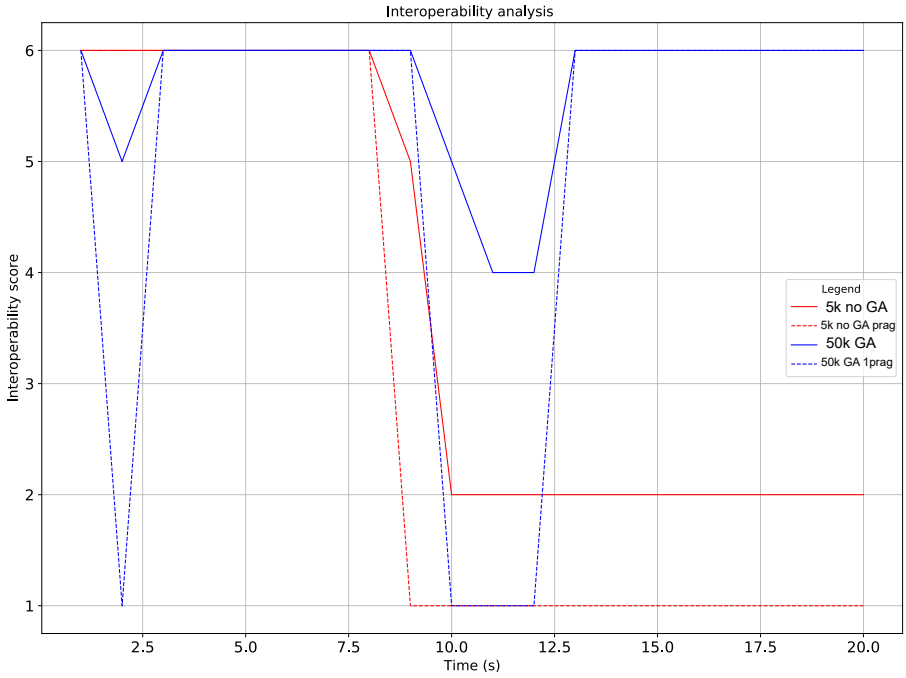
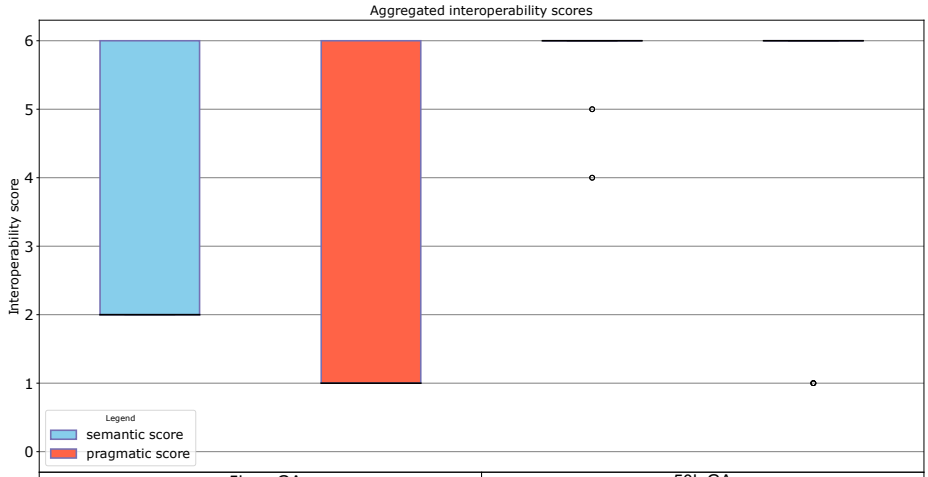


Figure 7-15: Interoperability analysis for scenario S_4 - XCS comparison including SI (solid lines) and PI scores (dashed lines)



	5k no GA	50k GA
Median_semantic	2.0	6.0
Median_pragmatic	1.0	6.0
Mean_semantic	3.75	5.7
Mean_pragmatic	3.0	5.0
Std_semantic	1.95	0.64
Std_pragmatic	2.45	2.0

Figure 7-16: Aggregated representation of the SI and PI scores for scenario S_4 XCS comparison

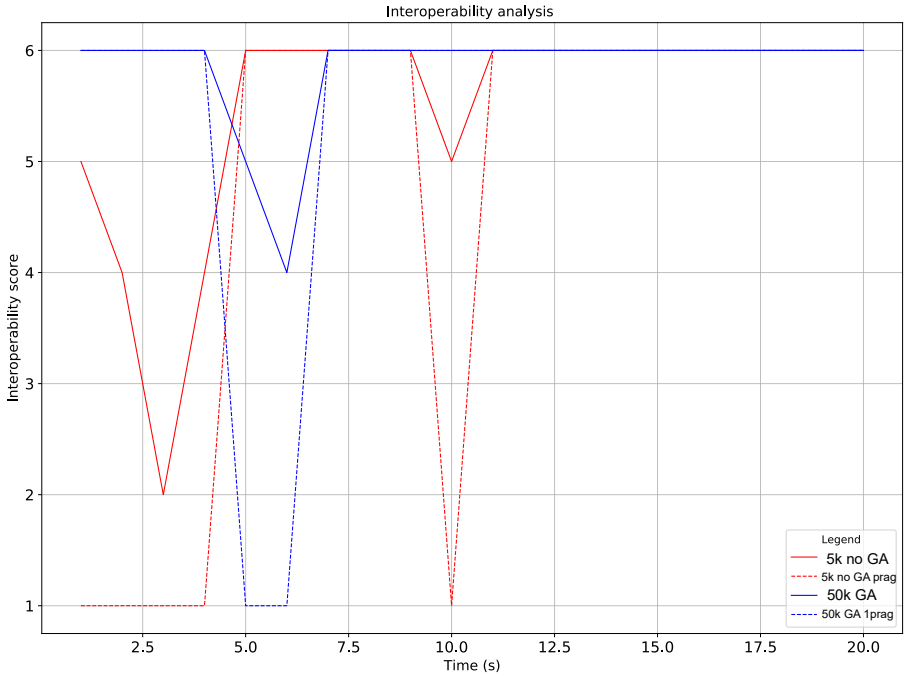
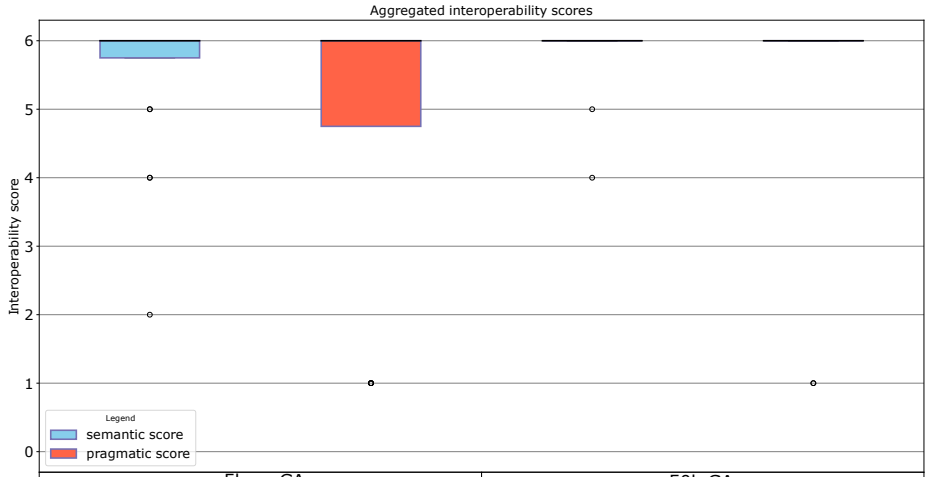


Figure 7-17: Second run of interoperability analysis for scenario S_4 - XCS comparison including SI (solid lines) and PI scores (dashed lines)



	5k no GA	50k GA
Median_semantic	6.0	6.0
Median_pragmatic	6.0	6.0
Mean_semantic	5.5	5.85
Mean_pragmatic	4.75	5.5
Std_semantic	1.02	0.48
Std_pragmatic	2.17	1.5

Figure 7-18: Aggregated representation of the SI and PI scores for the second run of scenario S_4 XCS comparison

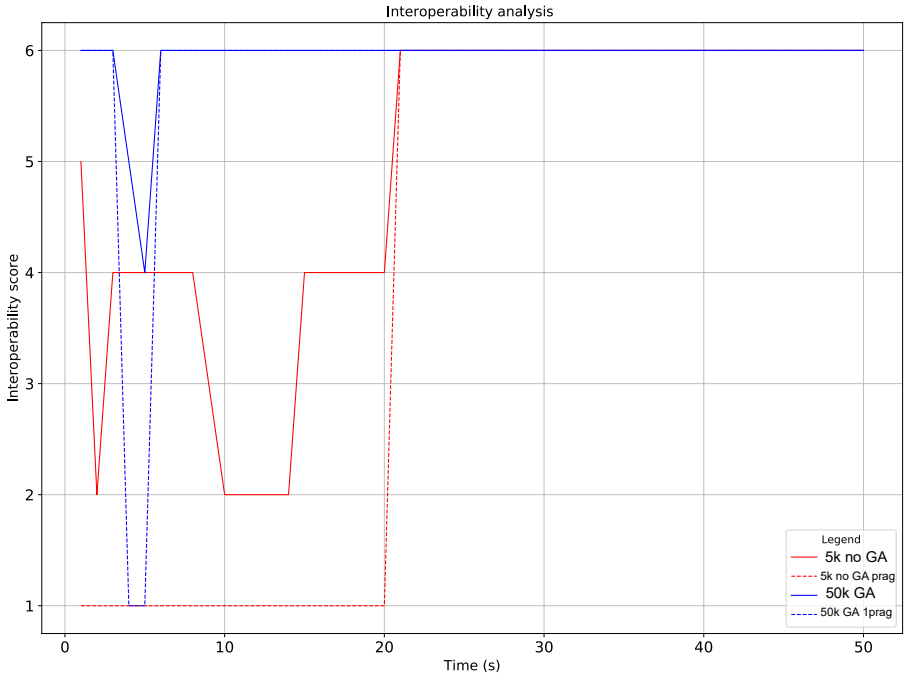
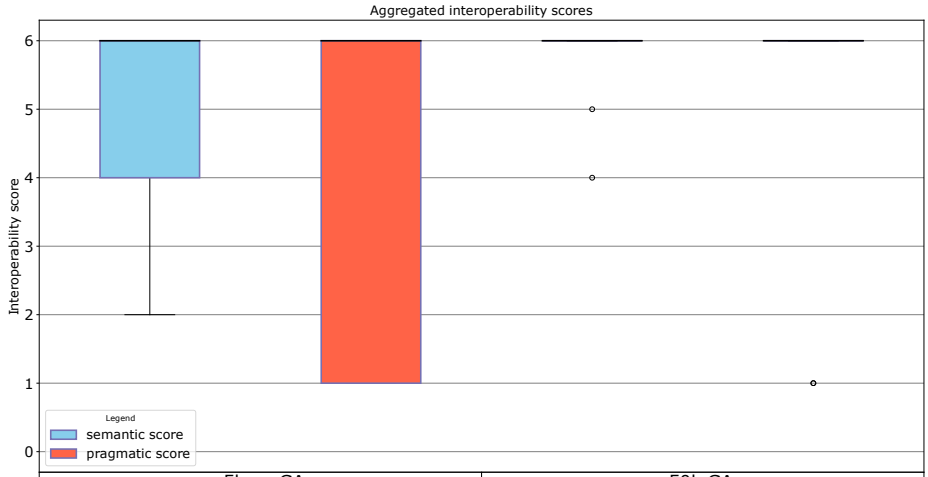


Figure 7-19: Third run of interoperability analysis for scenario S_4 - XCS comparison including SI (solid lines) and PI scores (dashed lines)



	5k no GA	50k GA
Median_semantic	6.0	6.0
Median_pragmatic	6.0	6.0
Mean_semantic	4.96	5.94
Mean_pragmatic	4.0	5.8
Std_semantic	1.41	0.31
Std_pragmatic	2.45	0.98

Figure 7-20: Aggregated representation of the SI and PI scores for the third run of scenario S_4 - XCS comparison

Metric	Value
Execution time	7 m
Used memory (MB)	56
Number of sent messages	118
Mean agent execution time (seconds)	0,001208333
Number of actions performed	35

Table 7.6: Experiment summary for scenario S_4

The results of scenario S_4 in figure 7-13 indicate medium deviations in SI scores between iterations 4 and 9 and strong deviations of PI scores for all three configurations between iterations 4 and 11. In all configurations, the PI score deteriorates to the lowest level and improves again to the maximum score, with *expert RB* being the slowest to recover. Figure 7-14 features stable SI and PI scores with minor outliers. The metrics measured an execution time of 7 seconds, 118 messages sent between the agents and 35 actions performed. The mean agent execution time was 0,001208333 seconds.

The XCS comparison was performed in three subsequent simulation runs with each run featuring a different simulated randomness. In each run, two XCS configurations are compared, a GA-disabled configuration (*expert RB*) with 5000 iterations and a GA-enabled configuration (XCS 50k) with 50.000 iterations. Results in 7-15 and 7-16 display a generally lower interoperability score for *expert RB* with a mean SI and PI score of 3.75 and 3.0 compared to 5.7 and 5.0 for XCS 50k. The interoperability analysis shows a drop in SI and PI scores for *expert RB* after iteration 8 which are not recovered. In comparison, in XCS 50k the SI and PI scores drop as well after iteration 8, but recover to the maximum interoperability level after iteration 13. *expert RB* generally experiences a worse performance than XCS 50k as evident in all three test runs. In the interoperability analysis results of figure 7-17 and figure 7-19 we note that XCS 50k is quicker to recover the maximum interoperability score than *expert RB*.

7.3.5 Scenario S_F results

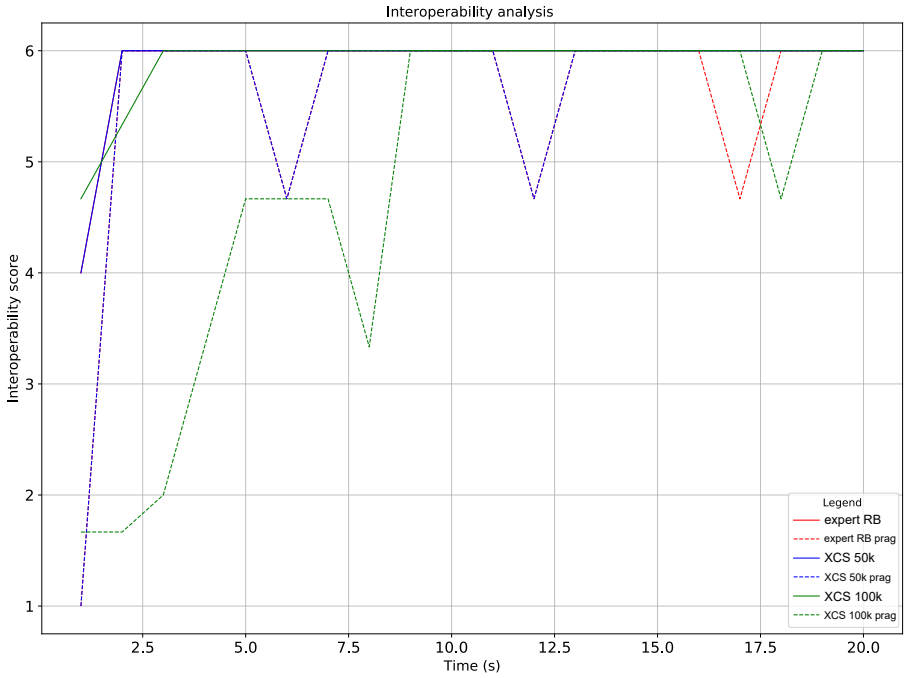
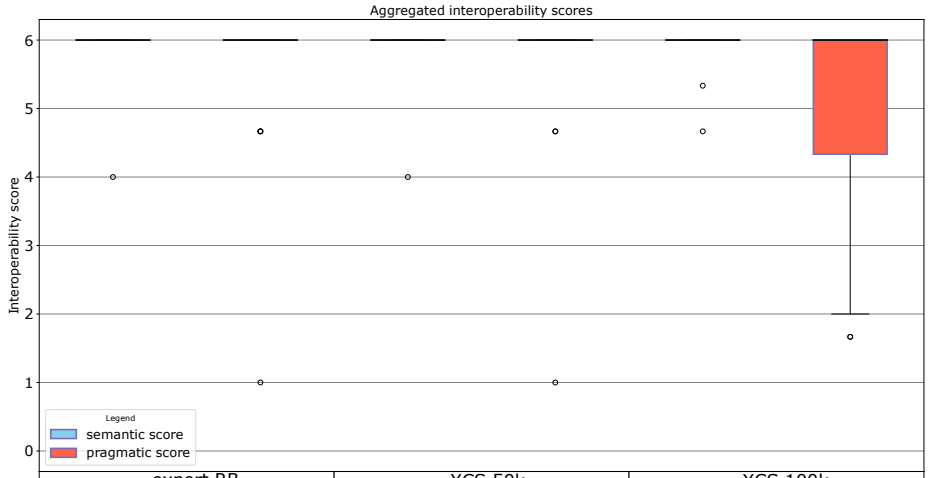


Figure 7-21: Interoperability analysis for scenario S_F including SI (solid lines) and PI scores (dashed lines)



	expert RB	XCS 50k	XCS 100k
Median_semantic	6.0	6.0	6.0
Median_pragmatic	6.0	6.0	6.0
Mean_semantic	5.9	5.9	5.9
Mean_pragmatic	5.55	5.62	4.83
Std_semantic	0.44	0.44	0.32
Std_pragmatic	1.15	1.13	1.54

Figure 7-22: Aggregated representation of the SI and PI scores for scenario S_F

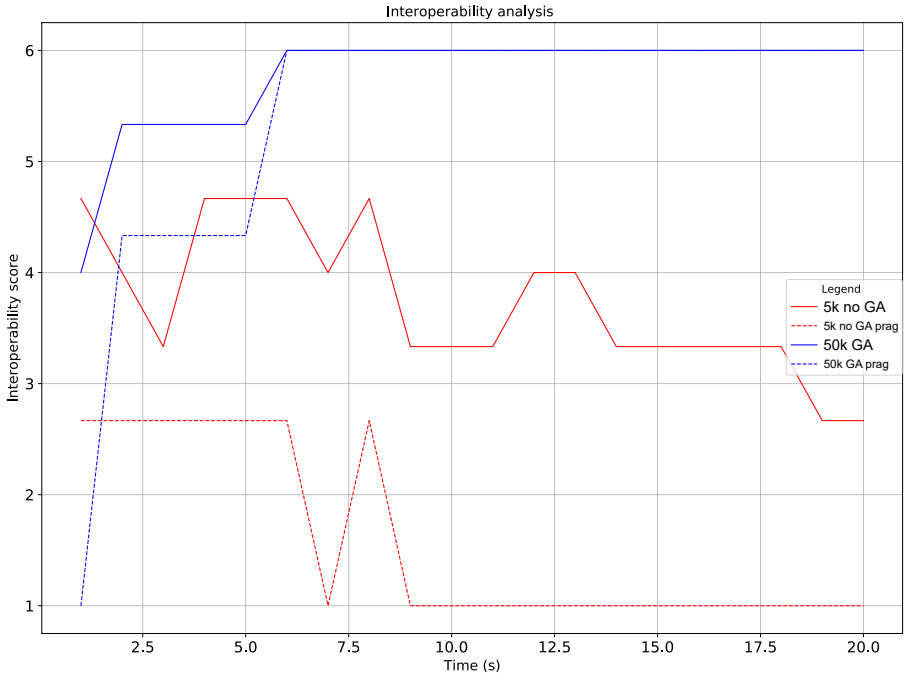
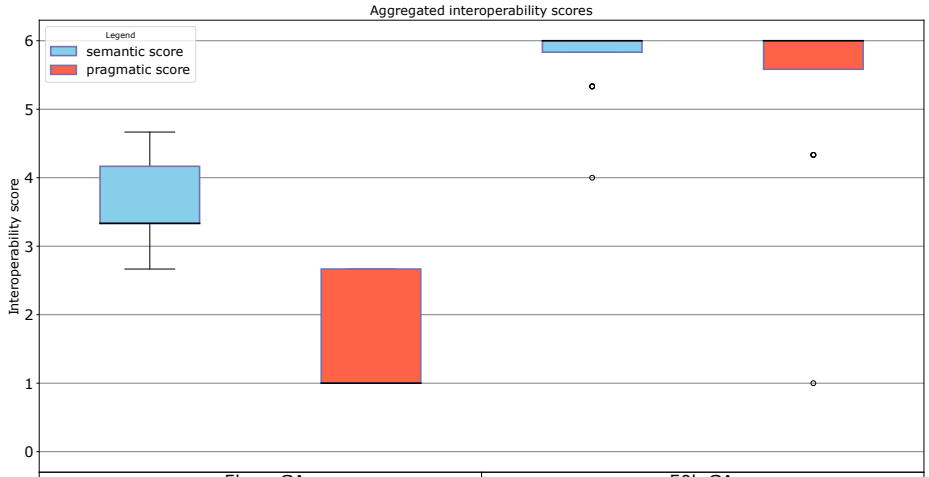


Figure 7-23: Interoperability analysis for scenario S_F - XCS comparison including SI (solid lines) and PI scores (dashed lines)



	5k no GA	50k GA
Median_semantic	3.33	6.0
Median_pragmatic	1.0	6.0
Mean_semantic	3.73	5.77
Mean_pragmatic	1.58	5.42
Std_semantic	0.65	0.48
Std_pragmatic	0.79	1.21

Figure 7-24: Aggregated representation of the SI and PI scores for scenario S_F - XCS comparison

Metric	Value
Execution time	18 m
Used memory (MB)	56
Number of sent messages	1062
Mean agent execution time (seconds)	0,145368272
Number of actions performed	165

Table 7.7: Experiment summary for scenario S_F

The interoperability analysis for scenario S_F evaluates different starting configurations. In figure 7-21, only the expert rule-system I-IOP agent is tested while in figure 7-23, the same two XCS configurations (5k no-GA (*expert RB*), 50k GA(ID_2)) as in the case of S_4 are compared. Figure 7-21 shows that *expert RB* and $XCS\ 50k$ achieved a solid interoperability performance, with relatively constant interoperability scores - verified in figure 7-22 as well. $XCS\ 100k$ requires 8 iterations to reach the maximum PI score, which is reflected in figure 7-22 with the lowest overall mean PI score. Also in all tested configurations, short PI interruptions at iteration 6,11,16 and 18 can be observed which in all cases recovered to the maximum PI score after one additional iteration.
The XCS comparison results (figure 7-23 and 7-24) highlight a deteriorating interoperability performance for $5k\ no\ GA$ from an initial interoperability score of (4.5,2.5) to (2.5,0). On the contrary, configuration $50k\ GA$ achieved maximum interoperability scores after iteration 5. Figure 7-24 displays significantly higher mean semantic and pragmatic scores for $50k\ GA$ compared to $5k\ no\ GA$.

7.3.6 Scenario S_P results

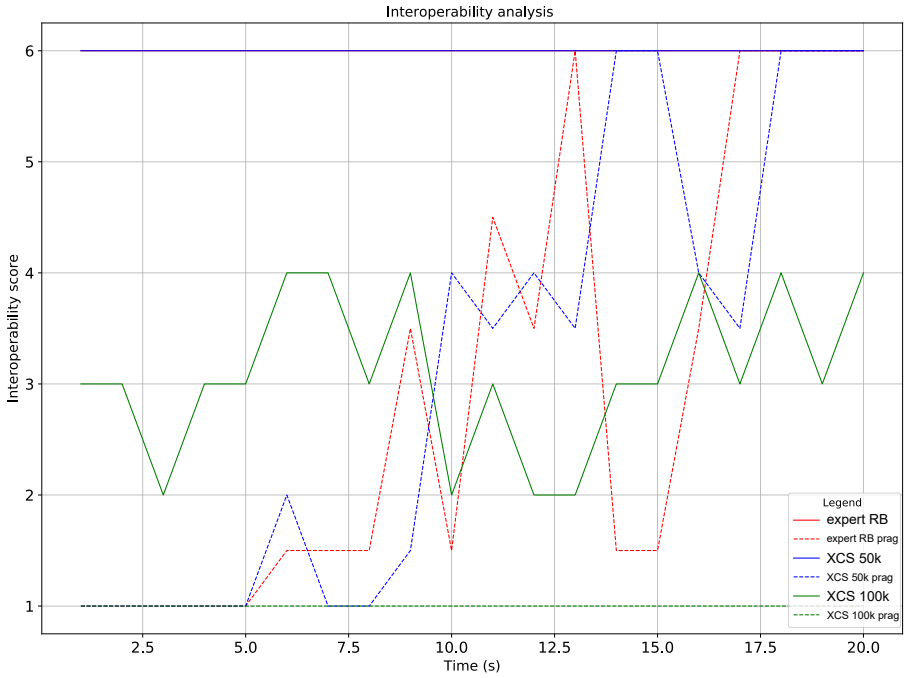
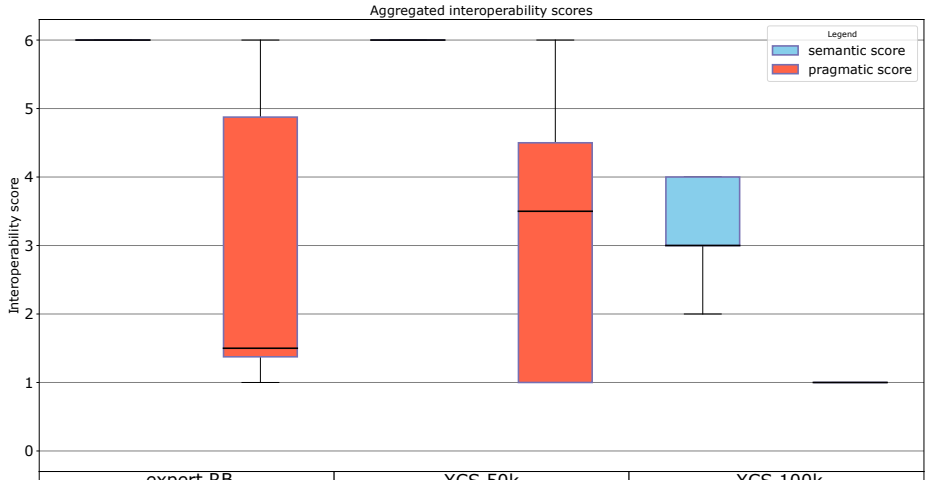


Figure 7-25: Interoperability analysis for scenario S_P including SI (solid lines) and PI scores (dashed lines)



	expert RB	XCS 50k	XCS 100k
Median_semantic	6.0	6.0	3.0
Median_pragmatic	1.5	3.5	1.0
Mean_semantic	6.0	6.0	3.1
Mean_pragmatic	2.95	3.15	1.0
Std_semantic	0.0	0.0	0.7
Std_pragmatic	2.02	1.99	0.0

Figure 7-26: Aggregated representation of the SI and PI scores for scenario S_P

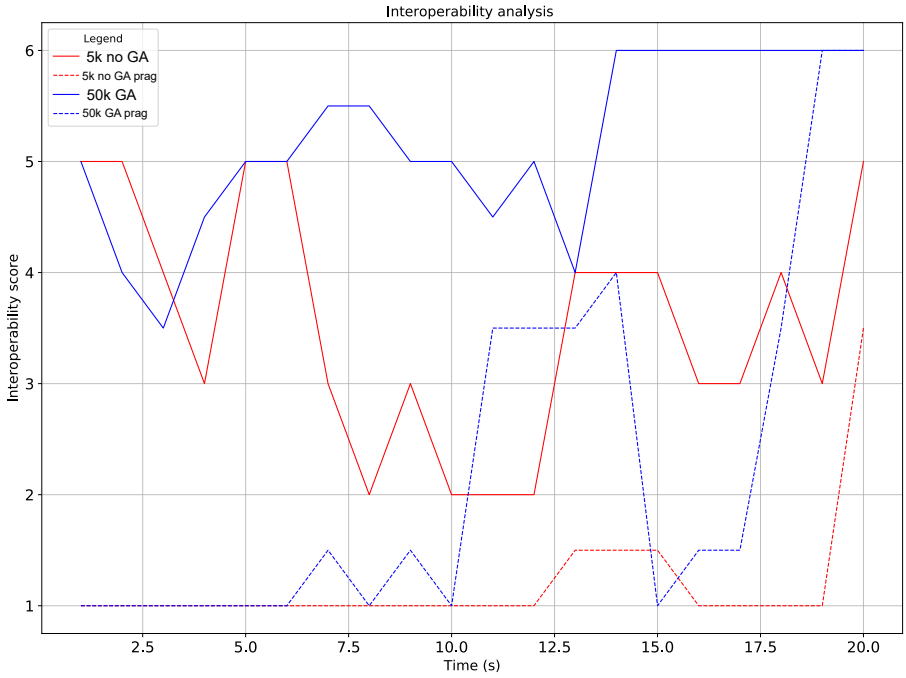
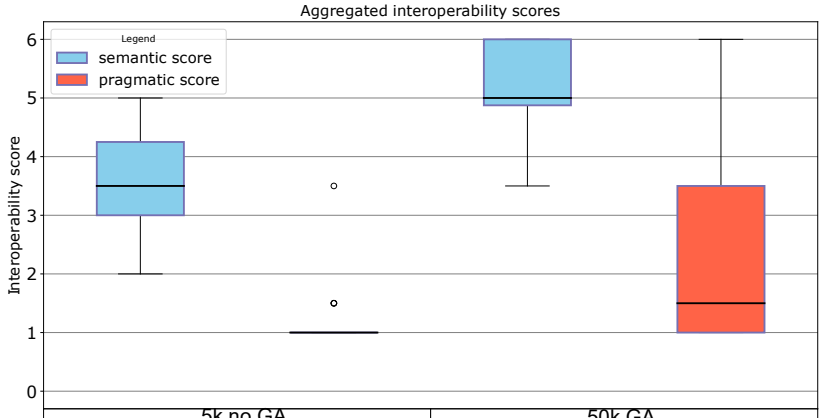


Figure 7-27: Interoperability analysis for scenario S_P - XCS comparison including SI (solid lines) and PI scores (dashed lines)



	5k no GA	50k GA
Median_semantic	3.5	5.0
Median_pragmatic	1.0	1.5
Mean_semantic	3.55	5.18
Mean_pragmatic	1.2	2.25
Std_semantic	1.07	0.76
Std_pragmatic	0.56	1.64

Figure 7-28: Aggregated representation of the SI and PI scores for scenario S_P - XCS comparison

Metric	Value
Execution time	12 m
Used memory (MB)	56
Number of sent messages	470
Mean agent execution time (seconds)	0,039586305
Number of actions performed	531

Table 7.8: Experiment summary for scenario S_P

Finally, in scenario S_P, specifically the pragmatic interoperability level was evaluated to demonstrate the bounds of the I-IOP abilities to maximize interoperability at runtime. Looking at the figure 7-26, XCS 50k marks the best overall performance with a median PI score of 3.5. *expert RB* and XCS 100k only achieved a median PI score of 1.5 and 1. The SI scores reach maximum levels in *expert RB* and XCS 50k while the median SI score in XCS 100k levels at 3. Figure 7-25 furthermore shows, that while *expert RB* and XCS 50k eventually reach the maximum PI level after 17 iterations, XCS 100k cannot achieve an improvement in the PI scores and the SI scores remain around 3 for the complete simulation.
The XCS comparison for scenario S_P present a low PI and medium SI score of 50k *GA* until iteration 12.5 when SI reaches the maximum level and iteration 18 when PI follows. Similar to previous scenarios, configuration 5k *no GA* experiences a worse interoperability performance which manifests in a mean SI score of 3.44 and mean PI score of 1.2 and a maximum SI and PI score of 5 and 3.5.

7.3.7 XCS training performance

To evaluate the training performance of the XCS learning module, multiple XCS configurations are compared. In particular, the DSC and DSP agents were trained using the XCS configurations found in section A.4.2 which differentiate between the number of trained iterations and an active or inactive genetic algorithm. The aim behind this experiment is to compare the advantage of the genetic algorithm of XCS in scenarios S_4, S_F and S_P and hence the overall advantage of using XCS vs a standard reinforcement learning procedure. Additionally, by evaluating multiple XCS configurations the best trade off between training time and learning performance can be evaluated. This is an important metric since as of the agent's requirement to gain new knowledge at runtime and thus will need to be retrained regularly. If this re-training however takes up a significant amount

of time, it could hurt the overall agent's usefulness. To present the training results, the situation, action, fitness, timestamp, average reward, reward prediction error, experience, action set size and numerosity returned by the training model are listed, where the timestamp specifies the last time the classifier was part of a GA competition, experience refers to the number of applied parameter updates, action set size estimates the moving average of the action sets the classifier was part of, and numerosity specifies the number of (micro-) classifiers, this macro-classifier actually represents [But15]. The agents were trained for 1000, 5000, 10.000, 25.000, 50.000 and 100.000 iterations. These numbers were chosen deliberately to observe a noteworthy effect on the training performance, since a gap of only 10.000 iterations each did not deliver meaningful results in previous testing results.

The summarized XCS training results are given in tables A.1 and A.2. The training time for the DSC and DSP agents lies between 11 and 237 seconds from lowest to highest number of training iterations, respectively 11 to 240 seconds for the DSP agent. We observe a steady increase in training times with higher numbers of trained iterations. Considering the difference between active and non-active GA, we observe, that for all experiments with a disabled GA, the agent was able to learn highly accurate classifiers which is evidenced by the average fitness over all no-GA configurations roughly around 0.99 . Also the number of learned classifiers is considerably higher compared to an enabled GA. However, one has to keep in mind that these classifiers are not generalized, which means that, opposed to the classifiers in the GA configurations, they only match their respective situations. The best training results with enabled GA were obtained by the configuration XCS5k for the DSC and XCS25k for the DSP agents with an average fitness of 0.73143 and 0.912648 respectively. In general, the fitness scores in case of an enabled GA are lower for the DSC results compared to the DSP results. The lowest fitness scores are found with configuration XCS1k for both agents. The highest number of learned situations was achieved by XCS50k - no GA for the DSC agent and XCS5k - no GA for the DSP agent. Notably, in case of the DSC agent, the number of learned situations decreases with higher numbers of iterations while it increases in case of the DSC agent results.

7.4 Summary

This chapter described a feasibility study to evaluate the I-IOP agent architecture in a simulated IoT ecosystem. The results are shown for different interoperability scenarios and use cases. The next chapter will discuss these results and present their implications within the runtime interoperability context and conclude on the performance of the I-IOP agent architecture, if it is able to fulfill the runtime interoperability requirements.

Chapter 8

Discussion

8.1 Aim

After the conceptual design and empirical evaluation of the I-IOP architecture, the final chapter will revisit the research questions and discuss how the architecture contributes to the improvement of interoperability between IoT platforms.

8.2 Revisiting the research questions

To investigate the current state of the art of interoperability in the IoT, a thorough literature survey was performed. The background search revealed that there are two main streams of approaches: standardization and software based solutions. The main issues to address with these approaches (as of the time of writing) are the high amount of proprietary solutions, diversity and heterogeneity of IoT ecosystems [SBKK17]. Interoperability research has spawned a plethora of semantic interoperability solutions and middleware architectures to solve the problem. In the context of the IoT ecosystem simulation and BIG IoT results the thesis has affirmed that these elements provide the foundation for interoperability across IoT domains and platforms. Syntactic interoperability can only be achieved through common exchange formats and semantic interoperability can only be reached through semantic technologies. The BIG IoT results have joined both elements into a unified architecture for easy adoption by IoT platform providers. Yet, the amount of existing solutions raised the point, why the interoperability problem is still not solved.

It was found that the foundational elements of syntax and semantics do not constitute a complete interoperability stack for IoT systems, as presented and argued during the introduction of

the runtime interoperability problem. As shown in chapter 5, the Internet of Things (IoT) requires (due to its complexity and heterogeneity) a new, runtime oriented perspective on interoperability. The requirements to establish runtime interoperability that were gathered accordingly cover the existing presently mentioned issues in literature, such as semantics and service orientation but add additional requirements that are not yet properly addressed. These requirements have been confirmed by joining the background on System of systems (SoS) research with a holistic, conceptual model on the interactions in IoT ecosystems which resulted in a thorough picture of the essential flows in IoT ecosystems. Such a holistic perspective is still underdeveloped in IoT literature but necessary to analyze interoperability in these systems from a practical-oriented and more complete perspective. Also, experimental studies on higher interoperability levels are rarely implemented making it difficult to assess and compare interoperability approaches. The mapping between existing approaches with the runtime interoperability requirements confirms that existing approaches only address a minority of the requirements that would be necessary for successful runtime interoperability. Essential requirements such as adaptivity, autonomy and learning have been identified as not sufficiently covered by existing solutions which is the reason for the remainder of open interoperability problems, especially when it comes to dynamic and highly uncertain environments. A potential reason for this could be a missing general view on how to properly model IoT ecosystems, especially across multiple domains.

Thus, to aid the process of creating holistic, runtime-oriented interoperability solutions, the conceptual model from chapter 4 and the interoperability lifecycle as an abstract representation of the core elements of interoperability, based on service collaboration in SoS environments have been introduced. Describing the interoperability problem from a multi-agent perspective (albeit existing research in MAS based middleware approaches) provides an ideal basis to model dynamic IoT ecosystems. As the simulator implementation has demonstrated, this agent-based model greatly eases the process of designing testbeds for IoT ecosystems. Through the conceptual model, all interoperability related information flows can be represented. The interoperability lifecycle is able to reproduce existing interoperability solutions and can thus help in identifying domain-specific interoperability problems. However, it is not geared to be used with very detailed, domain-specific requirements, for example in the manufacturing domain. In these cases, the model has to be extended to include additional constraints on the interoperation, e.g. in the form of communication constraints, industry standards, etc. One of the identified runtime requirements is the

ability for IoT systems to measure the state of interoperability. It was identified that a proper metric for this is missing from literature, rather interoperability is usually measured/assessed from different standpoints and using different metrics - depending on the domain. Usually this is a manual, design-time oriented approach. The presented semantic and pragmatic interoperability metrics are concrete, implementable metrics that cover core properties of semantic and pragmatic interoperability mismatches. The feasibility study and the result evaluation have shown that these metrics can be used equally well for a current assessment of interoperability but also to design utility functions and strategies for IoT agents. Comparing the metric to existing quantification approaches such as the i-Score 2, it is more straightforward to apply to concrete IoT related interoperability problems. Especially the pragmatic interoperability metric focuses on concrete, runtime-specific measurements while the semantic interoperability metric can be used in conjunction with ontology alignment processes.

8.3 Evaluation of the I-IOP agent

Introducing an optimization component in the runtime interoperability lifecycle effectively bridges the gap between existing design-time oriented interoperability solutions and future, autonomous and runtime-adaptive solutions. Without a way to interact with the underlying system, an IoT agent will not be able to execute interoperability related decisions. The literature survey has shown that, albeit the amount of work in MAS and MDP based problem statements, such a link has not been successfully established for the IoT. Therefore, the feasibility study addresses the question whether the developed I-IOP architecture is able to solve or improve the state of the art interoperability solutions and solve the problem of runtime interoperability, in particular the transaction optimization problem. Compared to existing approaches, the I-IOP agent is the first interoperability middleware that implements a semantic and pragmatic interoperability metric on a quantifiable, measurable level and provides a self-adaptive middleware component for autonomous adaptations at runtime. For the evaluation, the results of the feasibility study are discussed in the context of the previously defined runtime interoperability requirements. The underlying assumption states that if the runtime interoperability requirements are met, the I-IOP architecture effectively solves the runtime interoperability problem.

Analyzing the overall performance of the I-IOP agent in the feasibility study, the results first of all show, that in all scenarios the maximum SI and PI scores were achieved. Generally, we observe a trend that the pragmatic interoperability scores feature a larger standard deviation and generally lower interoperability scores compared to the semantic results. This could be due to the generally more complex adaptations necessary to improve pragmatic interoperability compared to semantic interoperability. The positive performance across the various scenarios confirms the domain-independent properties of the agent and its applicability across different use cases. The fact that the agent was able to achieve a maximization of interoperability scores despite the relatively short iteration cycles demonstrates the responsiveness and real-time nature of the architecture. In case of a hundred or more iterations to reach the maximum interoperability level this would have seriously questioned its usefulness in real-time critical applications, for example in the smart agriculture use case where real-time responsiveness is vital. The responsiveness is additionally affirmed by the quick reaction time (usually one or two iteration cycles) in case of an interoperability issue.

The service orientation and discoverability requirements are demonstrated by the design of the simulator experiments, since service oriented communication is the only means by which agents can exchange data. The results affirm that the service oriented architecture of the interoperability lifecycle implementation leads to the proper discovery of required services in each scenario and message exchange between the systems interfaces. No manual integration was performed within the I-IOP DSC system, i.e the systems autonomously enter into a business relationships based on their internal goals (since the goals drive the discovery mechanism). This is reflected in the simulation outcomes, as all scenarios require first of all that the DSC agent queries for available providers and negotiates a contract. In all results, the discovery was successful which is proven by a general interoperability level > 0 across experiments. Thus the architecture seems well suited for dynamic, open environments such as the one depicted in the environmental-aware routing service were it is expected that IoT service compositions change more frequently.

The adaptivity requirement is met as the I-IOP agents reliably respond to a deterioration of interoperability in all tested scenarios. Moreover the decisional autonomy of the agent is verified, as a manual intervention was not performed to maximize interoperability by the experiment operator. Rather, the agents were shown to be able to reason about the current level of interoper-

ability and act accordingly to improve interoperability. The interoperability curves show steady behaviour which means that the agents were able to analyze interoperability over all experienced states. Especially in scenario S_4 the advantages of an autonomous acting system become apparent due to the random DSP perturbations. As the graphs show, the agents kept the interoperability level relatively steady across the runtime of the experiment. This is especially valuable in use cases which require strong resilience guarantees, such as in the smart production case where perturbation related failures can have costly consequences.

In the S_2 results, we observe that the agent also properly addresses those cases where no intervention is required since the maximum interoperability level is already achieved. This further highlights the inter-workings of interoperability reasoning, measurement and autonomous self-adaptation inside the AC module. This is irrespective of the consideration of semantic or pragmatic interoperability. The simulation shows the effect of the adaptations on the semantic and pragmatic processes of the underlying transaction module. The simulated System under observation and controls in this case implement the semantic and pragmatic processes, discussed previously. Specifically the results of S_F and S_P indicate that in both scenarios the agent architecture was able to properly address the semantic and pragmatic interoperability requirements, by the general increase in semantic and pragmatic interoperability. Since both these use cases impose different requirements for interoperability, these results seem promising to further develop the I-IOP agent architecture towards usage in production systems. The results demonstrate that the AC module can address both interoperability metrics separately while the agent is only performing either a semantic or pragmatic interoperability related action at a time. This reduces unnecessary overhead and costs which is further important for the real-world applicability of the I-IOP agent. In the smart production related scenario S_P the advantages of the proposed architecture become further apparent, since the low pragmatic interoperability levels in the beginning of the simulation indicate that without proper adaptation the systems would not be able to interoperate. This motivates the importance and relevance of such automated processes to cope with interoperability issues especially in complex industrial settings were systems are generally harder to become interoperable.

On a side note, in the S_P scenario problems regarding the transaction optimization process are visible in the case of XCS_100k. Here, the optimization process gets stuck in a local optima around semantic interoperability level 3 - 4. This indicates that the agent's adaptation process is still sensitive to initial starting conditions. This illustrates the earlier assumption about the importance

of being aware of the variability of IoT ecosystems, i.e. that the IoT system designer needs to properly configure the agent to the respective context, since the same configuration for an industrial context might not work in a farming or smart city related context. Furthermore, a trend can be observed that pragmatic interoperability is more challenging to achieve than semantic interoperability, since the pragmatic scores usually lack behind the semantic scores. This can be explained by the fact that the pragmatic feature space offers more freedom for adaptation and is more challenging to navigate furthermore assigning more weight to the expertise of the IoT expert to properly configure the agent before deployment. Nevertheless, the I-IOP agents were also able to maximize pragmatic interoperability alongside semantic interoperability.

Considering the learning performance of the I-IOP agent's XCS learning module, the results affirm that the agent is able to learn the essential situation-action rules necessary for the tested scenarios. The results have shown that for both the GA-disabled and GA-enabled case the XCS module worked as expected. In case of a disabled GA, the learning performance is strong and reliable - exhibiting better performance than in the GA-enabled case. This observation can be explained since in the case of exact matchings, the classifiers can be fitted exactly to the situations which increases the fitness for each classifier. One has to keep in mind though that this always comes at the cost of a worse generalization performance. Since the number of learned situations does not increase significantly with increased training iterations over 5000, we can conclude that the disabled GA configuration could be applied for clearly arranged, rather static environments such as in the smart agriculture or smart production examples where generalization is not necessary as no new situations will be experienced at runtime by the agent and where fast training is more important.

Regarding the performance of the GA-enabled experiments we observe the following: On the one hand, the DSC and DSP agents were able to learn the correct actions for the presented training situations and the average fitness value of 0.71 and 0.89 shows that they are quite confident in their learned experience. Notably, the fitness between the DSC and DSP agent trainings differs which can be explained with the different action spaces and underlying strategies that were used for training. The agents already achieved good performance values after 5k iterations in case of the DSC agent and 10k iterations for the DSP agent, with only slightly improvements thereafter. On the other hand, the agents seem to struggle with situations in which not a single element of the situation identifier is essential but multiple. In other words, if an action decision depends on

multiple state features at once, the XCS module seems to experience problems during training. Another issue appeared when, in some cases, the agents seem to learn abstracted situations when instead they should be be conditioned on the concrete condition parameters. This problem seemed to resolve with increased training cycles though, again requiring a trade-off between training time and accuracy. Comparing the training time for each iteration with the gain in fitness, it is apparent that a larger amount of iterations achieves no significant improvement which would justify an increase in training time over two minutes. Probably, the agent overfits with increased number of iterations to the already learned situation-action mappings. Increasing the exploration ratio could be a solution to this problem, but this brings the disadvantage of longer training cycles which might or might not be desirable. We thus have to balance between completeness and real-time applicability of the agent's policy. In a smart city context it is better for an agent to have a rather complete set of situation knowledge since more unexpected situations can occur while in a smart farming scenario the real-time nature is more important, i.e. less training cycles since the amount of experienced situations might be smaller. To conclude, it seems to be generally better to use less iterations for the initial training, which might not capture all situations than to risk overfitting. Situations which are not captured during the initial training process can later be added in subsequent feedback cycles from the environment. This would also allow the agent to adapt the policy to highly specific environment situations as well. The rather low training time of around eighty seconds makes it possible to run the training procedure at runtime and constantly feed in new knowledge which makes it suitable for real-time use cases.

As a comment to the overall training results, it is obvious that the action space (see table A.3) is relatively small and furthermore very general and not use case specific. This is the result of a number of tests with different action spaces and action configurations. This final action set produced the best performance. It was observed during experimentation that the general learning performance decreased with increasing number of actions, probably because with increased situation and action spaces, the combinatorial number of situation-action mappings gets too large for the agent to learn in a reasonable amount of time. This requires the designer of the learning module to carefully decide, which situations and actions are relevant. This also means, that a general purpose situation-action map is not feasible in this architecture - hence training *generalist* agents which are applicable in multiple environments is difficult. This is in line with the earlier proposition that agents should be carefully configured and trained to the respective domain they are operating in.

Thus a more realistic assumption would be, that the AC module is trained differently, specialized on the underlying domain problem. This might contradict the underlying statement, that interoperability solutions in the IoT should be domain independent. But the general I-IOP architecture is in fact domain independent which means, in case of context switches, the AC-module can be exchanged, while the rest of the agent remains the same. For example, the same agent can be used in different smart factory installments, by being re-trained when switching contexts to learn the proper policy for each context. This also brings the advantage, that multiple AC-modules could be pre-trained before the agent starts operation and switched at runtime based on the need which increases the agent's responsiveness and improves resilience. In a smart production context we can envision a collection of pre-trained models which are exchanged based on the factory configuration and production needs. Another alternative is to switch from a discrete to a continuous action space which is better suited to handle large action spaces which could occur in a smart production scenario when IoT systems experience large degrees of configurability and parameters. However this would require a different learning module which is not considered in the scope of this thesis.

Breaking down the runtime performance of the trained agents, we need to consider the results of the comparison between expert policies vs XCS learned policies. Also, the XCS learning policies results which compare the difference between an active and inactive GA. The general assumption is that the XCS - learned policy works equally well or better than the expert rule-base policy. This is obviously a trivial baseline assumption which needs to be supported by the experimental results in oder to justify the learning module in the first place. The next assumption verifies whether the XCS50k policy indeed achieves an overall better performance compared to the XCS10k policy since the XCS training results have shown similar training results between the XCS50k and XCS10k training configurations. Based on the scenario results the first assumption generally holds, however in scenario S_P we notice that the expert policy achieved slightly better results. This could be due to the active exploratory component of the XCS agent. The exploratory component has the advantage, that the agent is able to react to new situations at runtime (as evidenced by the XCS comparison results), nevertheless this also means that in some situations the agent explores the action space instead of choosing a learned action. This could also explain the pragmatic interoperability results of XCS_100k in S_F which show a decrease in interoperability between episodes 6 and 8. This is again an indicator for the importance of proper configuration by an IoT expert. In case of an already ideal configuration, there is no need for the agent to ex-

plore better solutions hence the learned policies should be disabled. However, in case there was no expert configuration, the agent can improve performance through its exploratory, optimizing nature.

The second assumption does not hold, which is especially obvious in the case of the S_P results. If we compare the overall XCS performances across all scenarios, it is apparent that (except in S_1 and S_4) the XCS10k module has achieved the same or better performance than the XCS50k module. This is also visible in the boxplot information. The XCS10k module, although having learned fewer situations than the XCS50k module, shows an overall higher fitness for the learned situation action mappings in the training results. This means, as long as those situations for which the agent was not trained are not experienced at runtime, the adaptive performance will be better. In a real-world operation this requires careful consideration by the IoT expert to determine what levels of situational variability will be experienced by the agent to achieve best performance. The significantly worse performance by XCS50k in S_P probably has to be accounted to the earlier mentioned problem of local minima. Some results (for instance in figure 7-11) show oscillating interoperability scores. This could hint to problems with contradictory actions that the agent suggests between iterations. Such problems could be avoided through an action memory inside the agent that "blocks" actions for a certain time to avoid them being executed in circles. Also, the agent's learning model could be expanded to include the previous taken action alongside the situation information. Such improvements have not been considered yet and should be postponed for future enhancements of the I-IOP architecture.

Comparing the results of the distinct XCS comparison we want to analyze whether indeed the XCS agent with an enabled GA performs better at runtime than a GA-disabled, *classical* reinforcement learning agent. In all three evaluated scenarios we observe a trend that the GA-enabled configurations achieved better results, significantly even in the case of scenario S_F and S _4. Plausibly, the non generalizing behaviour keeps the agent's knowledge base limited to only those specific situations it has experienced during training. This is verified by the results since the GA-disabled agent gets stuck or experiences worsening interoperability performance over the course of the simulation, as shown in figure 7-23. The GA-enabled agent is faster to recover a deteriorated interoperability state and also more robust which is supported by the generally higher interoperability score in the boxplot aggregations. For an IoT system such as in smart production this means less failures and higher resilience. Also, the more the agent is able to adapt to new contexts the less manual effort is required thus significantly reducing costs and therefore increasing the overall

profitability of large scale IoT deployments. Furthermore, the assumption that the XCS agent is able to learn to react to new situations is backed up by the results as apparently, in case of missing situation-action mappings, the GA-disabled agent is not able to improve interoperability. This can be seen for example in figure 7-15 or figure 7-23. This is furthermore an assurance (for example for a farm manager or machine operator), that in critical situations which have not been thought of by an IoT expert, the system will remain operational. Summarizing, the XCS comparison shows that, although the disabled GA configurations achieved better training results, the online performance paints a different picture. The results in scenarios S_4, S_F and S_P therefore validate that the genetic algorithm in combination with a reinforcement learning component offers advantages over a pure reinforcement learning module which is an encouraging observation especially considering the properties of IoT ecosystems of self-organization,emergence and dynamic connectivity, i.e. frequent changes at runtime are to be expected. This underlines the fit of the I-IOP agent architecture to the core interoperability problem in IoT ecosystems and the need for runtime interoperability.

8.3.1 Effort and performance analysis

The measured metrics during experimentation highlight the general low footprint of the I-IOP architecture. Since larger amounts of training iterations do not lead to better performance during training, we assume training cycles between 10.000 and 50.000 iterations in operation. Therefore, the expected training time lies between roughly one or two minutes. Since a constant online retraining is not necessary, this keeps the training overhead reasonably low and thus applicable to a wide range of IoT scenarios (assuming a corresponding thread model so that training and agent execution can be parallelized). Thus, training delays should not be an issue, neither in end-consumer based use cases such as environmental-routing nor in business-to-business based industrial scenarios. As seen in the metric results of all simulated scenarios, the overall memory consumption of the agent is moderate at a couple of megabytes, which makes is possible to execute the agent also on constrained IoT platforms. This is highly relevant in industrial settings, were embedded platforms on industrial machines are predominant and allow to bring IoT solutions closed to the edge of computing. The rather large number of sent messages between the individual I-IOP agents points to the need for a stable network connection in a real-world deployment. This can generally be assumed to be the case though, especially in rather closed settings of smart farms or smart production lines. The mean agent execution time of 0,050962134 seconds supports the applicability of the I-IOP architecture for real-time scenarios as required especially, for example in the

smart production use case. Although the execution time will always vary (based on the underlying hardware) it does not exclude such real-time cases. Yet, we have to keep the general training time in mind, which requires parallelization in highly dynamic ecosystems to maintain real-time responsiveness.

Integration effort Although the real integration effort of the I-IOP architecture will only be visible when employed in *real-world* systems, the integration effort can be estimated through the feasibility study design. The fundamental agent architecture requires the IoT developer to design its IoT application/service logic accordingly. A plug-and-play solution is currently not feasible. Essentially this means that the developer needs to specify the search queries to be executed or the offerings to be registered. Also, the choice of provider systems after the output of the matchmaking functionality needs to be encoded by the developer according to the domain / customer requirements. The semantic embedding of messages can be assisted through code annotations, to help the semantic description of search queries, offerings and messages. This was also tested and verified in the BIG IoT project. Additionally, the interoperability metric (for evaluation of the interoperability situation) needs to be specified by the developer or customer of the service as well. This may seem like an unreasonable amount of effort but if we compare this one-time effort to the overall runtime gains we note that after the agent received its initial configuration, it can be used at runtime with no further input required from the developer until a change is desired.
The fact that the actions of the I-IOP agent only have an effect on the environment if implemented by the underlying SUOC means, that the biggest hurdle for integration lies in the implementation of the action space on the controlled system side. Essentially, this requires an initial effort from the IoT developer of the SUOC to implement the action space. Obviously, if the system does not support certain actions, the abilities of the I-IOP agent will be restricted. Conversely, the mightiness of the I-IOP agent rises simultaneously with the amount of implemented actions. In the feasibility study it was assumed that the IoT developer implements all actions of the action space - hence the results will differ if this is not the case. Due to this essential step for the success of the agent, the process of action implementation should be assisted by providing templates and solid documentation to help the IoT developer as much as possible. The simulation summaries indicate that the agents communicate a large number of messages, thus the IoT developer has to make sure that he secures his system from being overloaded by agent messages.

8.4 Summary

Table 8.1 gives an overview over all known runtime interoperability requirements (from section 5.3), and the evaluated level of achievement of the I-IOP agent based on the empirical results. The level of achievement is ranked from one '*' to the maximum of '***'.

Requirement/Approach	Level of achievement
Measurability	***
Discoverability	***
Service orientation	***
Semantic mechanisms	**
Pragmatic mechanisms	**
Adaptivity	***
Learning behaviour	**
Autonomy	**
Interoperability reasoning	***

Table 8.1: The level of achievement with respect to the runtime interoperability requirements that the I-IOP agent achieves.

Overall it was shown, that the I-IOP agent architecture successfully implements the runtime interoperability requirements. The runtime interoperability in IoT ecosystems is defined as achieved when IoT systems are able to autonomously fulfill a common mission (or goal), for a non-negligible amount of time. Reflecting on the overall results we observe that in all scenarios the I-IOP agents were able to collaboratively achieve sufficient, or optimal levels of interoperability. The adaptivity and autonomy requirements are fully met through the I-IOP architecture and also the learning behaviour of the AC module shows promising results, although there is still room for improvements, particular in cases of local optima. It is further shown that the IoT expert who is responsible to instantiate and oversee the deployment is an important resource to make the I-IOP agents achieve optimal performance. This means, the human factor still plays an important role in the increasing automated world. Nevertheless, the I-IOP agent architecture presents a valuable improvement over state-of-the-art IoT middleware which, as of yet, has not considered the application of machine learning as part of the middleware offering but rather focused on the levels of service orientation and semantics (with some exceptions). In addition, comparing the outcomes of the feasibility study with the other approaches for interoperability (compare table 5.7), it is evident that especially with regards to measurability, learning behaviour and autonomy this thesis adds a valuable contribution.

As a disclaimer, we should be aware that the results were obtained through a *simulation* of an IoT ecosystem. Even though the results were not derived from a real-world ecosystem, they nevertheless show a strong indicator for the relevance of the I-IOP architecture to achieve runtime interoperability. The simulator was designed based on real-world testbeds and careful literature analysis but a validation was made difficult due to missing real-world environments for testing as of the time of writing. Since the simulation results are promising and considering the increasing development efforts in the IoT, a subsequent step will be to test the architecture in a real-world IoT system. This task will be postponed to potential follow-up work of this thesis. It will be interesting to compare the architecture in different real-world setting which would further generate valuable research data for additional improvements. Also, the correctness of the implementation of the simulation was tested using unit tests. However it can not be excluded that inaccuracy in the implementation (especially with regards to the simulation of transactions between DSC and DSP agents) also affect the interoperability scores.

Part IV

Conclusion

Chapter 9

Conclusion & Outlook

Interoperability was shown to be a thorny issue in the context of the Internet of Things, especially with regards to ecosystem formation and establishment of IoT marketplaces. As we have seen, tremendous research efforts have been invested already but still there are many open questions left. This thesis lines up with this stream of research to analyze the problem of interoperability between IoT platforms, which is one of the main issues preventing the IoT from reaching its expected value. A conceptualization of an IoT ecosystem from a System of systems (SoS) perspective was presented, followed by a formalism of the runtime interoperability problem which conceptualizes interoperability not from the usual design time perspective but from an Organic Computing inspired runtime-oriented viewpoint. This helped to analyze the deficits of existing interoperability methods followed by a proposal for a self-adaptive IoT agent architecture which addresses the runtime interoperability problem on a technical level and allows to adapt an interoperability functionality at runtime to optimize semantic and pragmatic interoperability. The formalization of runtime interoperability addresses all levels of the interoperability lifecycle and serves as a central guideline to cover the major aspects of the interoperability problem in the IoT, specifically pragmatic interoperability, which has not yet been previously handled on an architectural level. Also, the introduced formalization elaborate interoperability metrics for semantic and pragmatic interoperability in order to make the interoperability problem quantifiable and measurable, which is an active problem in current research. The I-IOP agent model was evaluated in a simulated IoT environment, designed based on common IoT use cases. An evaluation with existing work in the design of IoT middleware was provided as well, to show the benefits of the introduced runtime oriented perspective. The agent architecture lays a solid foundation for handling the interoperabil-

ity problem in the IoT from a new perspective which should provide better customer experience and more resilience between distributed systems. The architecture offers a regulatory feedback mechanism through observing the current state between the SUOC and adapting interoperability relevant parameters accordingly. It is realized through a rule-based online learning system, in this case through a XCS system. Such dynamic approaches, when introduced into real IoT systems, are desired to cross IoT platform boundaries to enable the future vision of the Internet of Things (IoT 2.0) as complex SoS where systems of different scales and capabilities are connected in cross-domain, cross-platform manner to improve human life and to enable new business areas and technology advances.

9.1 Future work

To further verify the conceptual model and the interoperability lifecycle, additional domain scenarios and use cases should be evaluated, for example using the simulator or in real-world testbeds. This would advance the validation of the underlying assumptions and could also extend the model with additional domain specifics, if needed. However, as a note of caution, the model should still be kept at a rather abstract level to avoid the problem of making it too domain dependent. The semantic and pragmatic interoperability metrics are not assumed to be complete, but instead they present ground for further extensions and evaluations in concrete IoT systems. Further technical improvements could well make it possible to monitor the complete communication scenario of agent communication. The local optima problem should be addressed by testing other optimization algorithms and re-evaluate the presented scenarios. Also, one could experiment with continuous state and action spaces to make the AC module more versatile and robust. Generally there are also other topics relevant in the general scope of the IoT interoperability problem, such as privacy, trust and security. Although these topics are not addressed in the scope of this thesis, they should be linked with the I-IOP architecture to show how the architecture influences these issues. Another research direction could lead into the game theory territory. In this thesis, only the collaborative settings between IoT systems are considered but one could include confrontational/adversarial settings and analyze the effect on the interoperability level. This could be an interesting analysis regarding the benefits/disadvantages of cooperation vs. competition in an open IoT ecosystem. With increased platform connectivity by different vendors and therefore increased decentralization of the IoT infrastructure this will be an important topic in the near future.

Appendix A

Appendix

Acronyms

API Application programming interface.

Bluetooth SIG Bluetooth Special interest group.

CC Cloud computing.

CMP Connectivity management platform.

CS Constituent system.

DLT Distributed ledger technology.

DSC Digital service consumer.

DSE Digital service ecosystem.

DSP Digital service provider.

EU European union.

IaaS Infrastructure as a service.

IEEE Institute of Electrical and Electronics Engineers.

INSP Infrastructure provider.

IoT Internet of Things.

J2EE Java 2 enterprise edition.

JSON-LD JSON Linked data.

LCIM Levels of conceptual interoperability.

LCS Learning classifier system.

LISI Levels of Information Systems Interoperability.

LORA Low range wide area network.

M2M Machine to machine.

MMEI Maturity model for Enterprise interoperability.

MQTT Message Queuing Telemetry Transport.

NFC Near field communication.

OCA Observer Controller Architecture.

OWL Web Ontology Language.

OWL-S Web Ontology Language for Web Services.

P2P Peer-to-peer.

PaaS Platform as a service.

PI Pragmatic interoperability.

QoS Quality of service.

REST Representational state transfer.

RFID Radio-frequency identification.

RPC Remote Procedure Call.

SaaS Software as a service.

SADP self-aware systems design properties.

SDK Software development kit.

SI Semantic interoperability.

SLA Service level agreement.

SOA Service oriented architecture.

SoS System of systems.

SSN Semantic sensor network.

SUOC System under observation and control.

UI User interface.

URI Uniform resource identifier.

VM Virtual machine.

W3C World Wide Web Consortium.

WoT Web of Trust.

A.1 Detailed XCS training results - DSC

A.1.1 XCS 1k - no GA

```
000000010111 => 8  [0.95602]
000000010111 => 8
    Time Stamp: 44
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9560199746936111
    Experience: 14
    Action Set Size: 1.0
    Numerosity: 1
000000101101 => 8  [0.93128]
000000101101 => 8
    Time Stamp: 40
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9312812104587673
    Experience: 12
    Action Set Size: 1.0
    Numerosity: 1
000000111000 => 8  [0.94502]
000000111000 => 8
    Time Stamp: 273
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9450249683670139
    Experience: 13
    Action Set Size: 1.0
    Numerosity: 1
000000101000 => 17  [0.89263]
000000101000 => 17
    Time Stamp: 139
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.892626891341824
    Experience: 10
    Action Set Size: 1.0
    Numerosity: 1
000000011011 => 8  [0.91410]
000000011011 => 8
    Time Stamp: 162
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
```

A.1.2 XCS 1k

```
00#0#0#0#01# => 17  [0.57562]
00#0#0#0#01# => 17
    Time Stamp: 1000
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.5756178367019197
    Experience: 18
    Action Set Size: 41.02589602299904
    Numerosity: 4
```

A.1.3 XCS 5k - no GA

```
000000111010 => 8  [0.91410]
000000111010 => 8
    Time Stamp: 3921
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
000000011001 => 8  [0.99998]
000000011001 => 8
    Time Stamp: 64
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999977699477809
    Experience: 48
    Action Set Size: 1.0
    Numerosity: 1
000000011100 => 8  [0.99998]
000000011100 => 8
    Time Stamp: 275
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999977699477809
    Experience: 48
    Action Set Size: 1.0
    Numerosity: 1
000000110100 => 8  [0.99997]
000000110100 => 8
    Time Stamp: 125
    Average Reward: 1000.0
    Error: 0.0
```

```
    Fitness: 0.9999721243472612
    Experience: 47
    Action Set Size: 1.0
    Numerosity: 1
000000100110 => 8  [0.99998]
000000100110 => 8
    Time Stamp: 13
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999977699477809
    Experience: 48
    Action Set Size: 1.0
    Numerosity: 1
000000111100 => 8  [0.99987]
000000111100 => 8
    Time Stamp: 88
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998670785296495
    Experience: 40
    Action Set Size: 1.0
    Numerosity: 1
000000101000 => 17  [0.99996]
000000101000 => 17
    Time Stamp: 651
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999564442925956
    Experience: 45
    Action Set Size: 1.0
    Numerosity: 1
000000001100 => 8  [0.99993]
000000001100 => 8
    Time Stamp: 35
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999319442071806
    Experience: 43
    Action Set Size: 1.0
    Numerosity: 1
000000010101 => 8  [0.94502]
000000010101 => 8
    Time Stamp: 4065
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9450249683670139
    Experience: 13
    Action Set Size: 1.0
    Numerosity: 1
000000000011 => 17  [0.99998]
000000000011 => 17
    Time Stamp: 134
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999821595822472
    Experience: 49
    Action Set Size: 1.0
    Numerosity: 1
000000100111 => 8  [0.99995]
000000100111 => 8
    Time Stamp: 343
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999455553657445
    Experience: 44
    Action Set Size: 1.0
    Numerosity: 1
000000100100 => 8  [0.99921]
000000100100 => 8
    Time Stamp: 109
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9992077262976735
    Experience: 32
    Action Set Size: 1.0
    Numerosity: 1
000000101100 => 8  [0.99991]
000000101100 => 8
    Time Stamp: 40
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999149302589757
    Experience: 42
    Action Set Size: 1.0
    Numerosity: 1
000000111101 => 8  [0.99998]
000000111101 => 8
    Time Stamp: 261
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999821595822472
    Experience: 49
    Action Set Size: 1.0
    Numerosity: 1
000000010011 => 8  [0.99989]
000000010011 => 8
    Time Stamp: 92
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000001101 => 8  [0.99959]
000000001101 => 8
    Time Stamp: 302
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9995943558644088
    Experience: 35
    Action Set Size: 1.0
    Numerosity: 1
000000111011 => 8  [0.99991]
000000111011 => 8
    Time Stamp: 59
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999149302589757
    Experience: 42
    Action Set Size: 1.0
    Numerosity: 1
000000001011 => 17  [0.99997]
```

```
000000001011 => 17
    Time Stamp: 61
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999651554340765
    Experience: 46
    Action Set Size: 1.0
    Numerosity: 1
000000001111 => 8  [0.99979]
000000001111 => 8
    Time Stamp: 65
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997923102025773
    Experience: 38
    Action Set Size: 1.0
    Numerosity: 1
000000110101 => 8  [0.99999]
000000110101 => 8
    Time Stamp: 70
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999885821326382
    Experience: 51
    Action Set Size: 1.0
    Numerosity: 1
000000110001 => 8  [0.99949]
000000110001 => 8
    Time Stamp: 600
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999492944830511
    Experience: 34
    Action Set Size: 1.0
    Numerosity: 1
000000101001 => 17  [0.99968]
000000101001 => 17
    Time Stamp: 464
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999675484691527
    Experience: 36
    Action Set Size: 1.0
    Numerosity: 1
000000111000 => 8  [0.99968]
000000111000 => 8
    Time Stamp: 77
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999675484691527
    Experience: 36
    Action Set Size: 1.0
    Numerosity: 1
000000011011 => 8  [0.99991]
000000011011 => 8
    Time Stamp: 136
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999149302589757
    Experience: 42
    Action Set Size: 1.0
    Numerosity: 1
000000011110 => 8  [0.99993]
000000011110 => 8
    Time Stamp: 510
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999319442071806
    Experience: 43
    Action Set Size: 1.0
    Numerosity: 1
000000001010 => 17  [0.99991]
000000001010 => 17
    Time Stamp: 91
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999149302589757
    Experience: 42
    Action Set Size: 1.0
    Numerosity: 1
000000100000 => 17  [0.99921]
000000100000 => 17
    Time Stamp: 94
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9992077262976735
    Experience: 32
    Action Set Size: 1.0
    Numerosity: 1
000000011010 => 8  [0.99974]
000000011010 => 8
    Time Stamp: 814
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997403877532216
    Experience: 37
    Action Set Size: 1.0
    Numerosity: 1
000000000100 => 8  [0.99997]
000000000100 => 8
    Time Stamp: 390
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999721243472612
    Experience: 47
    Action Set Size: 1.0
    Numerosity: 1
000000101101 => 8  [0.99993]
000000101101 => 8
    Time Stamp: 1049
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999319442071806
    Experience: 43
    Action Set Size: 1.0
    Numerosity: 1
000000000101 => 8  [0.99949]
000000000101 => 8
    Time Stamp: 411
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999492944830511
```

```
    Experience: 34
    Action Set Size: 1.0
    Numerosity: 1
000000000110 => 8  [0.99974]
000000000110 => 8
    Time Stamp: 485
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997403877532216
    Experience: 37
    Action Set Size: 1.0
    Numerosity: 1
000000000001 => 17  [0.99974]
000000000001 => 17
    Time Stamp: 324
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997403877532216
    Experience: 37
    Action Set Size: 1.0
    Numerosity: 1
000000100001 => 17  [0.99983]
000000100001 => 17
    Time Stamp: 421
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998338481620619
    Experience: 39
    Action Set Size: 1.0
    Numerosity: 1
000000010111 => 8  [0.99989]
000000010111 => 8
    Time Stamp: 351
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000101110 => 8  [0.99983]
000000101110 => 8
    Time Stamp: 151
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998338481620619
    Experience: 39
    Action Set Size: 1.0
    Numerosity: 1
000000011101 => 8  [0.99876]
000000011101 => 8
    Time Stamp: 588
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9987620723401149
    Experience: 30
    Action Set Size: 1.0
    Numerosity: 1
000000111111 => 8  [0.99989]
000000111111 => 8
    Time Stamp: 563
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000101011 => 17  [0.99993]
000000101011 => 17
    Time Stamp: 298
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999319442071806
    Experience: 43
    Action Set Size: 1.0
    Numerosity: 1
000000111110 => 8  [0.99989]
000000111110 => 8
    Time Stamp: 197
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000011111 => 8  [0.99983]
000000011111 => 8
    Time Stamp: 493
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998338481620619
    Experience: 39
    Action Set Size: 1.0
    Numerosity: 1
000000010000 => 8  [0.99921]
000000010000 => 8
    Time Stamp: 312
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9992077262976735
    Experience: 32
    Action Set Size: 1.0
    Numerosity: 1
000000001001 => 17  [0.99974]
000000001001 => 17
    Time Stamp: 254
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997403877532216
    Experience: 37
    Action Set Size: 1.0
    Numerosity: 1
000000101111 => 8  [0.91410]
000000101111 => 8
    Time Stamp: 3858
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
000000110000 => 8  [0.99983]
000000110000 => 8
```

```
    Time Stamp: 356
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998338481620619
    Experience: 39
    Action Set Size: 1.0
    Numerosity: 1
000000110111 => 8  [0.99901]
000000110111 => 8
    Time Stamp: 409
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999009657872092
    Experience: 31
    Action Set Size: 1.0
    Numerosity: 1
000000010110 => 8  [0.99997]
000000010110 => 8
    Time Stamp: 830
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999651554340765
    Experience: 46
    Action Set Size: 1.0
    Numerosity: 1
000000100101 => 8  [0.99968]
000000100101 => 8
    Time Stamp: 419
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999675484691527
    Experience: 36
    Action Set Size: 1.0
    Numerosity: 1
000000000111 => 8  [0.97185]
000000000111 => 8
    Time Stamp: 3880
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.971852783803911
    Experience: 16
    Action Set Size: 1.0
    Numerosity: 1
```

A.1.4 XCS 5k

```
11#00####### => 4  [0.52009]
11#00####### => 4
    Time Stamp: 4971
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.5200867695339794
    Experience: 52
    Action Set Size: 46.86336080052103
    Numerosity: 23
01########## => 4  [0.82710]
01########## => 4
    Time Stamp: 4990
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.8271033953114357
    Experience: 287
    Action Set Size: 110.1872388751312
    Numerosity: 85
#0#0###0#0## => 17  [0.73881]
#0#0###0#0## => 17
    Time Stamp: 4979
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.7388116463209025
    Experience: 409
    Action Set Size: 97.35320625342044
    Numerosity: 70
1##1######## => 6  [0.83972]
1##1######## => 6
    Time Stamp: 4976
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.839723074186975
    Experience: 147
    Action Set Size: 94.87775834183924
    Numerosity: 80
```

A.1.5 XCS 10k - no GA

```
000000000100 => 8  [0.99921]
000000000100 => 8
    Time Stamp: 6165
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9992077262976735
    Experience: 32
    Action Set Size: 1.0
    Numerosity: 1
000000000000 => 17  [0.99758]
000000000000 => 17
    Time Stamp: 6779
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.997582172539287
    Experience: 27
    Action Set Size: 1.0
    Numerosity: 1
000000101100 => 8  [1.00000]
000000101100 => 8
```

```
    Time Stamp: 335
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999997943139872
    Experience: 69
    Action Set Size: 1.0
    Numerosity: 1
000000010111 => 8  [0.99921]
000000010111 => 8
    Time Stamp: 5054
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9992077262976735
    Experience: 32
    Action Set Size: 1.0
    Numerosity: 1
000000111001 => 8  [0.99921]
000000111001 => 8
    Time Stamp: 6533
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9992077262976735
    Experience: 32
    Action Set Size: 1.0
    Numerosity: 1
000000011110 => 8  [1.00000]
000000011110 => 8
    Time Stamp: 793
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999823317061
    Experience: 80
    Action Set Size: 1.0
    Numerosity: 1
000000101001 => 17  [0.99262]
000000101001 => 17
    Time Stamp: 8196
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9926213761574924
    Experience: 22
    Action Set Size: 1.0
    Numerosity: 1
000000000001 => 17  [0.99876]
000000000001 => 17
    Time Stamp: 6377
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9987620723401149
    Experience: 30
    Action Set Size: 1.0
    Numerosity: 1
000000001000 => 17  [0.95602]
000000001000 => 17
    Time Stamp: 8117
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9560199746936111
    Experience: 14
    Action Set Size: 1.0
    Numerosity: 1
000000001011 => 17  [0.97748]
000000001011 => 17
    Time Stamp: 8882
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000000110 => 8  [1.00000]
000000000110 => 8
    Time Stamp: 40
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999942104535
    Experience: 85
    Action Set Size: 1.0
    Numerosity: 1
000000110101 => 8  [0.89263]
000000110101 => 8
    Time Stamp: 8661
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.892626891341824
    Experience: 10
    Action Set Size: 1.0
    Numerosity: 1
000000111101 => 8  [0.98559]
000000111101 => 8
    Time Stamp: 8252
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9855886253076024
    Experience: 19
    Action Set Size: 1.0
    Numerosity: 1
000000111100 => 8  [0.98847]
000000111100 => 8
    Time Stamp: 7782
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9884709002460819
    Experience: 20
    Action Set Size: 1.0
    Numerosity: 1
000000100100 => 8  [0.94502]
000000100100 => 8
    Time Stamp: 8812
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9450249683670139
    Experience: 13
    Action Set Size: 1.0
    Numerosity: 1
000000110100 => 8  [1.00000]
000000110100 => 8
    Time Stamp: 97
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999999165639
    Experience: 104
```

```
    Action Set Size: 1.0
    Numerosity: 1
000000110011 => 8  [0.89263]
000000110011 => 8
    Time Stamp: 8753
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.892626891341824
    Experience: 10
    Action Set Size: 1.0
    Numerosity: 1
000000011001 => 8  [1.00000]
000000011001 => 8
    Time Stamp: 569
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999886922919
    Experience: 82
    Action Set Size: 1.0
    Numerosity: 1
000000110111 => 8  [0.93128]
000000110111 => 8
    Time Stamp: 8806
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9312812104587673
    Experience: 12
    Action Set Size: 1.0
    Numerosity: 1
000000001001 => 17  [0.91410]
000000001001 => 17
    Time Stamp: 8616
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
000000111010 => 8  [0.99991]
000000111010 => 8
    Time Stamp: 6173
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999149302589757
    Experience: 42
    Action Set Size: 1.0
    Numerosity: 1
000000100101 => 8  [1.00000]
000000100101 => 8
    Time Stamp: 91
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999981028814
    Experience: 90
    Action Set Size: 1.0
    Numerosity: 1
000000010101 => 8  [0.99993]
000000010101 => 8
    Time Stamp: 5325
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999319442071806
    Experience: 43
    Action Set Size: 1.0
    Numerosity: 1
000000101110 => 8  [1.00000]
000000101110 => 8
    Time Stamp: 496
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999998370387
    Experience: 101
    Action Set Size: 1.0
    Numerosity: 1
000000010110 => 8  [0.99807]
000000010110 => 8
    Time Stamp: 6657
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9980657380314296
    Experience: 28
    Action Set Size: 1.0
    Numerosity: 1
000000011111 => 8  [1.00000]
000000011111 => 8
    Time Stamp: 128
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999993783522
    Experience: 95
    Action Set Size: 1.0
    Numerosity: 1
000000011010 => 8  [0.99901]
000000011010 => 8
    Time Stamp: 6919
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999009657872092
    Experience: 31
    Action Set Size: 1.0
    Numerosity: 1
000000111011 => 8  [0.99979]
000000111011 => 8
    Time Stamp: 5077
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997923102025773
    Experience: 38
    Action Set Size: 1.0
    Numerosity: 1
000000100010 => 17  [0.99845]
000000100010 => 17
    Time Stamp: 6560
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9984525904251437
    Experience: 29
    Action Set Size: 1.0
    Numerosity: 1
000000101011 => 17  [1.00000]
000000101011 => 17
    Time Stamp: 509
```

```
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999998946887614
    Experience: 72
    Action Set Size: 1.0
    Numerosity: 1
000000011101 => 8  [0.98847]
000000011101 => 8
    Time Stamp: 7658
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9884709002460819
    Experience: 20
    Action Set Size: 1.0
    Numerosity: 1
000000001111 => 8  [1.00000]
000000001111 => 8
    Time Stamp: 4255
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999962585932229
    Experience: 56
    Action Set Size: 1.0
    Numerosity: 1
000000110001 => 8  [0.99996]
000000110001 => 8
    Time Stamp: 5029
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999564442925956
    Experience: 45
    Action Set Size: 1.0
    Numerosity: 1
000000010100 => 8  [0.99807]
000000010100 => 8
    Time Stamp: 6191
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9980657380314296
    Experience: 28
    Action Set Size: 1.0
    Numerosity: 1
000000011000 => 8  [0.96482]
000000011000 => 8
    Time Stamp: 7554
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9648159797548889
    Experience: 15
    Action Set Size: 1.0
    Numerosity: 1
000000001010 => 17  [1.00000]
000000001010 => 17
    Time Stamp: 288
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999909538335
    Experience: 83
    Action Set Size: 1.0
    Numerosity: 1
000000000111 => 8  [0.98847]
000000000111 => 8
    Time Stamp: 6970
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9884709002460819
    Experience: 20
    Action Set Size: 1.0
    Numerosity: 1
000000000010 => 17  [0.93128]
000000000010 => 17
    Time Stamp: 8411
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9312812104587673
    Experience: 12
    Action Set Size: 1.0
    Numerosity: 1
000000100001 => 17  [0.89263]
000000100001 => 17
    Time Stamp: 8787
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.892626891341824
    Experience: 10
    Action Set Size: 1.0
    Numerosity: 1
000000000011 => 17  [0.94502]
000000000011 => 17
    Time Stamp: 8210
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9450249683670139
    Experience: 13
    Action Set Size: 1.0
    Numerosity: 1
000000110010 => 8  [0.99622]
000000110010 => 8
    Time Stamp: 6912
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9962221445926361
    Experience: 25
    Action Set Size: 1.0
    Numerosity: 1
```

A.1.6 XCS 10k

```
#1#0######## => 4  [0.75711]
#1#0######## => 4
    Time Stamp: 9990
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.7571121424618339
```

```
    Experience: 592
    Action Set Size: 128.52287162400862
    Numerosity: 83
00#####1#### => 8  [0.50870]
00#####1#### => 8
    Time Stamp: 10000
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.5086996038162261
    Experience: 1805
    Action Set Size: 144.4581460843055
    Numerosity: 70
01########## => 4  [0.82342]
01########## => 4
    Time Stamp: 9978
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.8234152775749028
    Experience: 509
    Action Set Size: 143.68820478884786
    Numerosity: 106
1##1######## => 6  [0.63969]
1##1######## => 6
    Time Stamp: 9999
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.6396888477982674
    Experience: 316
    Action Set Size: 147.38659783393763
    Numerosity: 89
```

A.1.7 XCS 25k - no GA

```
000000101100 => 8  [0.99807]
000000101100 => 8
    Time Stamp: 21809
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9980657380314296
    Experience: 28
    Action Set Size: 1.0
    Numerosity: 1
000000111111 => 8  [1.00000]
000000111111 => 8
    Time Stamp: 9005
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999999999971
    Experience: 150
    Action Set Size: 1.0
    Numerosity: 1
000000101011 => 17  [0.99698]
000000101011 => 17
    Time Stamp: 21381
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9969777156741089
    Experience: 26
    Action Set Size: 1.0
    Numerosity: 1
000000110101 => 8  [1.00000]
000000110101 => 8
    Time Stamp: 16520
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999997428924841
    Experience: 68
    Action Set Size: 1.0
    Numerosity: 1
000000110010 => 8  [0.99989]
000000110010 => 8
    Time Stamp: 20586
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000110000 => 8  [0.99262]
000000110000 => 8
    Time Stamp: 22303
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9926213761574924
    Experience: 22
    Action Set Size: 1.0
    Numerosity: 1
000000100100 => 8  [1.00000]
000000100100 => 8
    Time Stamp: 17515
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999723932907
    Experience: 78
    Action Set Size: 1.0
    Numerosity: 1
000000101001 => 17  [0.99758]
000000101001 => 17
    Time Stamp: 21908
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.997582172539287
    Experience: 27
    Action Set Size: 1.0
    Numerosity: 1
000000010100 => 8  [0.99993]
000000010100 => 8
    Time Stamp: 19798
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999319442071806
    Experience: 43
    Action Set Size: 1.0
    Numerosity: 1
000000100110 => 8  [0.99937]
```

```
000000100110 => 8
    Time Stamp: 19946
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9993661810381388
    Experience: 33
    Action Set Size: 1.0
    Numerosity: 1
000000100001 => 17  [0.94502]
000000100001 => 17
    Time Stamp: 22804
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9450249683670139
    Experience: 13
    Action Set Size: 1.0
    Numerosity: 1
000000101000 => 17  [0.99999]
000000101000 => 17
    Time Stamp: 19244
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999908657061105
    Experience: 52
    Action Set Size: 1.0
    Numerosity: 1
000000000010 => 17  [0.98199]
000000000010 => 17
    Time Stamp: 23604
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.981985781634503
    Experience: 18
    Action Set Size: 1.0
    Numerosity: 1
000000000000 => 17  [0.98199]
000000000000 => 17
    Time Stamp: 23584
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.981985781634503
    Experience: 18
    Action Set Size: 1.0
    Numerosity: 1
000000010001 => 8  [1.00000]
000000010001 => 8
    Time Stamp: 14496
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999997962984
    Experience: 100
    Action Set Size: 1.0
    Numerosity: 1
000000001100 => 8  [0.99622]
000000001100 => 8
    Time Stamp: 21321
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9962221445926361
    Experience: 25
    Action Set Size: 1.0
    Numerosity: 1
000000001010 => 17  [0.99949]
000000001010 => 17
    Time Stamp: 21298
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999492944830511
    Experience: 34
    Action Set Size: 1.0
    Numerosity: 1
000000111000 => 8  [0.99845]
000000111000 => 8
    Time Stamp: 21248
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9984525904251437
    Experience: 29
    Action Set Size: 1.0
    Numerosity: 1
000000111110 => 8  [0.99262]
000000111110 => 8
    Time Stamp: 22639
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9926213761574924
    Experience: 22
    Action Set Size: 1.0
    Numerosity: 1
000000011100 => 8  [1.00000]
000000011100 => 8
    Time Stamp: 14449
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999996817163
    Experience: 98
    Action Set Size: 1.0
    Numerosity: 1
000000110001 => 8  [0.99996]
000000110001 => 8
    Time Stamp: 21356
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999564442925956
    Experience: 45
    Action Set Size: 1.0
    Numerosity: 1
000000011111 => 8  [0.89263]
000000011111 => 8
    Time Stamp: 24145
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.892626891341824
    Experience: 10
    Action Set Size: 1.0
    Numerosity: 1
000000001110 => 8  [1.00000]
000000001110 => 8
    Time Stamp: 16997
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999984675197842
```

```
    Experience: 60
    Action Set Size: 1.0
    Numerosity: 1
000000000110 => 8  [1.00000]
000000000110 => 8
    Time Stamp: 490
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999999999998
    Experience: 209
    Action Set Size: 1.0
    Numerosity: 1
000000100111 => 8  [0.99998]
000000100111 => 8
    Time Stamp: 19563
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999977699477809
    Experience: 48
    Action Set Size: 1.0
    Numerosity: 1
000000010000 => 8  [0.99999]
000000010000 => 8
    Time Stamp: 18985
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999857276657977
    Experience: 50
    Action Set Size: 1.0
    Numerosity: 1
000000110011 => 8  [0.98847]
000000110011 => 8
    Time Stamp: 22746
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9884709002460819
    Experience: 20
    Action Set Size: 1.0
    Numerosity: 1
000000011000 => 8  [0.99959]
000000011000 => 8
    Time Stamp: 19573
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9995943558644088
    Experience: 35
    Action Set Size: 1.0
    Numerosity: 1
000000001001 => 17  [0.99983]
000000001001 => 17
    Time Stamp: 20221
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998338481620619
    Experience: 39
    Action Set Size: 1.0
    Numerosity: 1
000000111101 => 8  [0.99528]
000000111101 => 8
    Time Stamp: 22841
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9952776807407951
    Experience: 24
    Action Set Size: 1.0
    Numerosity: 1
000000101010 => 17  [0.99901]
000000101010 => 17
    Time Stamp: 20916
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999009657872092
    Experience: 31
    Action Set Size: 1.0
    Numerosity: 1
000000011110 => 8  [0.99698]
000000011110 => 8
    Time Stamp: 21302
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9969777156741089
    Experience: 26
    Action Set Size: 1.0
    Numerosity: 1
000000100011 => 17  [0.99995]
000000100011 => 17
    Time Stamp: 20352
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999455553657445
    Experience: 44
    Action Set Size: 1.0
    Numerosity: 1
000000010011 => 8  [0.99622]
000000010011 => 8
    Time Stamp: 22007
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9962221445926361
    Experience: 25
    Action Set Size: 1.0
    Numerosity: 1
000000111010 => 8  [0.95602]
000000111010 => 8
    Time Stamp: 22420
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9560199746936111
    Experience: 14
    Action Set Size: 1.0
    Numerosity: 1
000000011010 => 8  [0.99845]
000000011010 => 8
    Time Stamp: 22123
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9984525904251437
    Experience: 29
    Action Set Size: 1.0
    Numerosity: 1
000000000001 => 17  [0.98559]
000000000001 => 17
```

```
    Time Stamp: 22832
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9855886253076024
    Experience: 19
    Action Set Size: 1.0
    Numerosity: 1
000000001000 => 17  [1.00000]
000000001000 => 17
    Time Stamp: 8827
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999999999998
    Experience: 174
    Action Set Size: 1.0
    Numerosity: 1
000000110100 => 8  [1.00000]
000000110100 => 8
    Time Stamp: 16062
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999998683609518
    Experience: 71
    Action Set Size: 1.0
    Numerosity: 1
000000101110 => 8  [0.99959]
000000101110 => 8
    Time Stamp: 22299
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9995943558644088
    Experience: 35
    Action Set Size: 1.0
    Numerosity: 1
000000010111 => 8  [0.89263]
000000010111 => 8
    Time Stamp: 23680
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.892626891341824
    Experience: 10
    Action Set Size: 1.0
    Numerosity: 1
000000000111 => 8  [0.98847]
000000000111 => 8
    Time Stamp: 22542
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9884709002460819
    Experience: 20
    Action Set Size: 1.0
    Numerosity: 1
000000011001 => 8  [0.94502]
000000011001 => 8
    Time Stamp: 22939
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9450249683670139
    Experience: 13
    Action Set Size: 1.0
    Numerosity: 1
000000100010 => 17  [0.99901]
000000100010 => 17
    Time Stamp: 21253
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999009657872092
    Experience: 31
    Action Set Size: 1.0
    Numerosity: 1
```

A.1.8 XCS 25k

```
#0#0###1#### => 8  [0.56259]
#0#0###1#### => 8
    Time Stamp: 24978
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.5625905431673819
    Experience: 4667
    Action Set Size: 154.24280391615952
    Numerosity: 78
#1#0######## => 4  [0.62985]
#1#0######## => 4
    Time Stamp: 24946
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.6298543578663202
    Experience: 1647
    Action Set Size: 180.67954605869596
    Numerosity: 98
1##1######## => 6  [0.98997]
1##1######## => 6
    Time Stamp: 24944
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9899747286296879
    Experience: 1532
    Action Set Size: 154.01592967565932
    Numerosity: 146
01########## => 4  [0.74992]
01########## => 4
    Time Stamp: 24946
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.749916905312802
    Experience: 1545
    Action Set Size: 171.21786277125264
    Numerosity: 113
00#####0#0## => 17  [0.57475]
00#####0#0## => 17
    Time Stamp: 24987
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.5747505208354617
```

```
    Experience: 2089
    Action Set Size: 154.31369800750562
    Numerosity: 83
00#######1## => 8  [0.51222]
00#######1## => 8
    Time Stamp: 25000
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.51222000724196
    Experience: 3876
    Action Set Size: 172.35855625266828
    Numerosity: 73
```

A.1.9 XCS 50k - no GA

```
000000011010 => 8  [0.99997]
000000011010 => 8
    Time Stamp: 44605
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999721243472612
    Experience: 47
    Action Set Size: 1.0
    Numerosity: 1
000000100110 => 8  [0.99998]
000000100110 => 8
    Time Stamp: 43419
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999821595822472
    Experience: 49
    Action Set Size: 1.0
    Numerosity: 1
000000101101 => 8  [0.99995]
000000101101 => 8
    Time Stamp: 44214
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999455553657445
    Experience: 44
    Action Set Size: 1.0
    Numerosity: 1
000000111011 => 8  [0.99995]
000000111011 => 8
    Time Stamp: 44506
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999455553657445
    Experience: 44
    Action Set Size: 1.0
    Numerosity: 1
000000101110 => 8  [0.98199]
000000101110 => 8
    Time Stamp: 48147
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.981985781634503
    Experience: 18
    Action Set Size: 1.0
    Numerosity: 1
000000000111 => 8  [1.00000]
000000000111 => 8
    Time Stamp: 43970
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999995982695064
    Experience: 66
    Action Set Size: 1.0
    Numerosity: 1
000000100101 => 8  [0.99622]
000000100101 => 8
    Time Stamp: 47290
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9962221445926361
    Experience: 25
    Action Set Size: 1.0
    Numerosity: 1
000000001110 => 8  [0.94502]
000000001110 => 8
    Time Stamp: 49120
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9450249683670139
    Experience: 13
    Action Set Size: 1.0
    Numerosity: 1
000000110011 => 8  [1.00000]
000000110011 => 8
    Time Stamp: 39091
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999997943139872
    Experience: 69
    Action Set Size: 1.0
    Numerosity: 1
000000100010 => 17  [0.99968]
000000100010 => 17
    Time Stamp: 45816
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999675484691527
    Experience: 36
    Action Set Size: 1.0
    Numerosity: 1
000000000000 => 17  [1.00000]
000000000000 => 17
    Time Stamp: 28458
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999999999998
    Experience: 169
    Action Set Size: 1.0
    Numerosity: 1
000000011011 => 8  [0.99949]
```

```
000000011011 => 8
    Time Stamp: 45410
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999492944830511
    Experience: 34
    Action Set Size: 1.0
    Numerosity: 1
000000111001 => 8  [0.93128]
000000111001 => 8
    Time Stamp: 49095
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9312812104587673
    Experience: 12
    Action Set Size: 1.0
    Numerosity: 1
000000111000 => 8  [0.99959]
000000111000 => 8
    Time Stamp: 46979
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9995943558644088
    Experience: 35
    Action Set Size: 1.0
    Numerosity: 1
000000011111 => 8  [0.94502]
000000011111 => 8
    Time Stamp: 48241
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9450249683670139
    Experience: 13
    Action Set Size: 1.0
    Numerosity: 1
000000000001 => 17  [0.99698]
000000000001 => 17
    Time Stamp: 47159
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9969777156741089
    Experience: 26
    Action Set Size: 1.0
    Numerosity: 1
000000001010 => 17  [0.96482]
000000001010 => 17
    Time Stamp: 48412
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9648159797548889
    Experience: 15
    Action Set Size: 1.0
    Numerosity: 1
000000000010 => 17  [0.99999]
000000000010 => 17
    Time Stamp: 42537
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999908657061105
    Experience: 52
    Action Set Size: 1.0
    Numerosity: 1
000000011000 => 8  [0.99999]
000000011000 => 8
    Time Stamp: 44509
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999857276657977
    Experience: 50
    Action Set Size: 1.0
    Numerosity: 1
000000000110 => 8  [0.98559]
000000000110 => 8
    Time Stamp: 48329
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9855886253076024
    Experience: 19
    Action Set Size: 1.0
    Numerosity: 1
000000011101 => 8  [0.99998]
000000011101 => 8
    Time Stamp: 45667
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999821595822472
    Experience: 49
    Action Set Size: 1.0
    Numerosity: 1
000000101111 => 8  [0.99262]
000000101111 => 8
    Time Stamp: 47216
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9926213761574924
    Experience: 22
    Action Set Size: 1.0
    Numerosity: 1
000000000100 => 8  [0.98559]
000000000100 => 8
    Time Stamp: 47520
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9855886253076024
    Experience: 19
    Action Set Size: 1.0
    Numerosity: 1
000000010001 => 8  [0.99807]
000000010001 => 8
    Time Stamp: 47106
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9980657380314296
    Experience: 28
    Action Set Size: 1.0
    Numerosity: 1
000000110001 => 8  [0.99949]
000000110001 => 8
    Time Stamp: 43901
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999492944830511
```

```
    Experience: 34
    Action Set Size: 1.0
    Numerosity: 1
000000001111 => 8  [0.99622]
000000001111 => 8
    Time Stamp: 47340
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9962221445926361
    Experience: 25
    Action Set Size: 1.0
    Numerosity: 1
000000101000 => 17  [1.00000]
000000101000 => 17
    Time Stamp: 44616
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999953232415286
    Experience: 55
    Action Set Size: 1.0
    Numerosity: 1
000000011100 => 8  [0.97748]
000000011100 => 8
    Time Stamp: 48280
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000001101 => 8  [0.99901]
000000001101 => 8
    Time Stamp: 46959
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999009657872092
    Experience: 31
    Action Set Size: 1.0
    Numerosity: 1
000000111111 => 8  [0.96482]
000000111111 => 8
    Time Stamp: 48175
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9648159797548889
    Experience: 15
    Action Set Size: 1.0
    Numerosity: 1
000000110010 => 8  [0.97748]
000000110010 => 8
    Time Stamp: 48654
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000111100 => 8  [0.99758]
000000111100 => 8
    Time Stamp: 46419
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.997582172539287
    Experience: 27
    Action Set Size: 1.0
    Numerosity: 1
000000001000 => 17  [0.99989]
000000001000 => 17
    Time Stamp: 45500
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000110110 => 8  [1.00000]
000000110110 => 8
    Time Stamp: 43087
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999962585932229
    Experience: 56
    Action Set Size: 1.0
    Numerosity: 1
000000101100 => 8  [0.99999]
000000101100 => 8
    Time Stamp: 45432
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999857276657977
    Experience: 50
    Action Set Size: 1.0
    Numerosity: 1
000000100011 => 17  [1.00000]
000000100011 => 17
    Time Stamp: 38086
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999996817163
    Experience: 98
    Action Set Size: 1.0
    Numerosity: 1
000000100000 => 17  [1.00000]
000000100000 => 17
    Time Stamp: 38724
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999976286018
    Experience: 89
    Action Set Size: 1.0
    Numerosity: 1
000000000101 => 8  [0.91410]
000000000101 => 8
    Time Stamp: 49124
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
000000111110 => 8  [0.91410]
000000111110 => 8
```

```
    Time Stamp: 48598
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
000000010010 => 8  [0.99995]
000000010010 => 8
    Time Stamp: 45413
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999455553657445
    Experience: 44
    Action Set Size: 1.0
    Numerosity: 1
000000001100 => 8  [0.99996]
000000001100 => 8
    Time Stamp: 44514
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999564442925956
    Experience: 45
    Action Set Size: 1.0
    Numerosity: 1
000000110101 => 8  [0.99078]
000000110101 => 8
    Time Stamp: 47388
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9907767201968655
    Experience: 21
    Action Set Size: 1.0
    Numerosity: 1
000000111010 => 8  [0.95602]
000000111010 => 8
    Time Stamp: 48559
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9560199746936111
    Experience: 14
    Action Set Size: 1.0
    Numerosity: 1
000000110100 => 8  [0.97748]
000000110100 => 8
    Time Stamp: 48658
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000100111 => 8  [0.96482]
000000100111 => 8
    Time Stamp: 48156
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9648159797548889
    Experience: 15
    Action Set Size: 1.0
    Numerosity: 1
000000010100 => 8  [0.99758]
000000010100 => 8
    Time Stamp: 47058
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.997582172539287
    Experience: 27
    Action Set Size: 1.0
    Numerosity: 1
000000010000 => 8  [0.94502]
000000010000 => 8
    Time Stamp: 48970
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9450249683670139
    Experience: 13
    Action Set Size: 1.0
    Numerosity: 1
000000011001 => 8  [0.99410]
000000011001 => 8
    Time Stamp: 47367
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9940971009259939
    Experience: 23
    Action Set Size: 1.0
    Numerosity: 1
000000001011 => 17  [0.99876]
000000001011 => 17
    Time Stamp: 46380
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9987620723401149
    Experience: 30
    Action Set Size: 1.0
    Numerosity: 1
000000100001 => 17  [0.99528]
000000100001 => 17
    Time Stamp: 48190
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9952776807407951
    Experience: 24
    Action Set Size: 1.0
    Numerosity: 1
```

A.1.10 XCS 50k

```
1##1######## => 6  [0.99510]
1##1######## => 6
    Time Stamp: 49968
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9950983093648633
```

```
    Experience: 3250
    Action Set Size: 154.64606536306866
    Numerosity: 152
01########## => 4  [0.73904]
01########## => 4
    Time Stamp: 49920
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.7390358468686428
    Experience: 3292
    Action Set Size: 177.07120871232786
    Numerosity: 119
#1#0######## => 4  [0.79651]
#1#0######## => 4
    Time Stamp: 49999
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.796508007766288
    Experience: 3169
    Action Set Size: 150.52918648756375
    Numerosity: 107
00#####0#0## => 17  [0.54468]
00#####0#0## => 17
    Time Stamp: 49981
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.5446777656705332
    Experience: 4674
    Action Set Size: 144.24183292583947
    Numerosity: 70
#0#0###0#0## => 17  [0.50259]
#0#0###0#0## => 17
    Time Stamp: 49956
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.5025859979247588
    Experience: 4274
    Action Set Size: 153.8515257232222
    Numerosity: 73
```

A.2 XCS training results - DSP

A.2.1 XCS 1k - no GA

None

A.2.2 XCS 1k

None

A.2.3 XCS 5k - no GA

```
000000001000 => 17  [0.99949]
000000001000 => 17
    Time Stamp: 1128
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999492944830511
    Experience: 34
    Action Set Size: 1.0
    Numerosity: 1
000000000101 => 12  [0.99989]
000000000101 => 12
    Time Stamp: 532
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000101010 => 12  [0.99999]
000000101010 => 12
    Time Stamp: 726
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999926925648884
    Experience: 53
    Action Set Size: 1.0
    Numerosity: 1
000000100100 => 12  [0.99997]
000000100100 => 12
    Time Stamp: 15
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999651554340765
    Experience: 46
    Action Set Size: 1.0
    Numerosity: 1
000000101000 => 11  [0.99983]
000000101000 => 11
    Time Stamp: 603
    Average Reward: 1000.0
```

```
    Error: 0.0
    Fitness: 0.9998338481620619
    Experience: 39
    Action Set Size: 1.0
    Numerosity: 1
000000111110 => 12  [0.99999]
000000111110 => 12
    Time Stamp: 151
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999941540519107
    Experience: 54
    Action Set Size: 1.0
    Numerosity: 1
000000011101 => 12  [0.99989]
000000011101 => 12
    Time Stamp: 322
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000101011 => 12  [0.99989]
000000101011 => 12
    Time Stamp: 41
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000001001 => 17  [0.99993]
000000001001 => 17
    Time Stamp: 543
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999319442071806
    Experience: 43
    Action Set Size: 1.0
    Numerosity: 1
000000100010 => 12  [0.99995]
000000100010 => 12
    Time Stamp: 219
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999455553657445
    Experience: 44
    Action Set Size: 1.0
    Numerosity: 1
000000110111 => 12  [0.96482]
000000110111 => 12
    Time Stamp: 3782
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9648159797548889
    Experience: 15
    Action Set Size: 1.0
    Numerosity: 1
000000010100 => 12  [0.99999]
000000010100 => 12
    Time Stamp: 157
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999908657061105
    Experience: 52
    Action Set Size: 1.0
    Numerosity: 1
000000111001 => 13  [0.99983]
000000111001 => 13
    Time Stamp: 1594
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998338481620619
    Experience: 39
    Action Set Size: 1.0
    Numerosity: 1
000000011001 => 13  [0.99959]
000000011001 => 13
    Time Stamp: 51
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9995943558644088
    Experience: 35
    Action Set Size: 1.0
    Numerosity: 1
000000110000 => 13  [0.99993]
000000110000 => 13
    Time Stamp: 400
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999319442071806
    Experience: 43
    Action Set Size: 1.0
    Numerosity: 1
000000101001 => 11  [0.99968]
000000101001 => 11
    Time Stamp: 263
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999675484691527
    Experience: 36
    Action Set Size: 1.0
    Numerosity: 1
000000101111 => 12  [0.99987]
000000101111 => 12
    Time Stamp: 61
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998670785296495
    Experience: 40
    Action Set Size: 1.0
    Numerosity: 1
000000110100 => 12  [0.99996]
000000110100 => 12
    Time Stamp: 218
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999564442925956
    Experience: 45
    Action Set Size: 1.0
    Numerosity: 1
```

```
000000010111 => 12  [0.99998]
000000010111 => 12
    Time Stamp: 89
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999977699477809
    Experience: 48
    Action Set Size: 1.0
    Numerosity: 1
000000010000 => 13  [0.99999]
000000010000 => 13
    Time Stamp: 580
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999908657061105
    Experience: 52
    Action Set Size: 1.0
    Numerosity: 1
000000100001 => 11  [0.99937]
000000100001 => 11
    Time Stamp: 425
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9993661810381388
    Experience: 33
    Action Set Size: 1.0
    Numerosity: 1
000000110101 => 12  [0.99949]
000000110101 => 12
    Time Stamp: 512
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999492944830511
    Experience: 34
    Action Set Size: 1.0
    Numerosity: 1
000000011110 => 12  [0.99921]
000000011110 => 12
    Time Stamp: 623
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9992077262976735
    Experience: 32
    Action Set Size: 1.0
    Numerosity: 1
000000100000 => 11  [0.99979]
000000100000 => 11
    Time Stamp: 347
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997923102025773
    Experience: 38
    Action Set Size: 1.0
    Numerosity: 1
000000100111 => 12  [0.99987]
000000100111 => 12
    Time Stamp: 534
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998670785296495
    Experience: 40
    Action Set Size: 1.0
    Numerosity: 1
000000011011 => 12  [0.99991]
000000011011 => 12
    Time Stamp: 648
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999149302589757
    Experience: 42
    Action Set Size: 1.0
    Numerosity: 1
000000001100 => 12  [0.99987]
000000001100 => 12
    Time Stamp: 230
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998670785296495
    Experience: 40
    Action Set Size: 1.0
    Numerosity: 1
000000101100 => 12  [0.99959]
000000101100 => 12
    Time Stamp: 271
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9995943558644088
    Experience: 35
    Action Set Size: 1.0
    Numerosity: 1
000000111111 => 12  [0.99974]
000000111111 => 12
    Time Stamp: 997
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997403877532216
    Experience: 37
    Action Set Size: 1.0
    Numerosity: 1
000000010101 => 12  [0.99999]
000000010101 => 12
    Time Stamp: 184
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999926925648884
    Experience: 53
    Action Set Size: 1.0
    Numerosity: 1
000000010001 => 13  [0.99807]
000000010001 => 13
    Time Stamp: 200
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9980657380314296
    Experience: 28
    Action Set Size: 1.0
    Numerosity: 1
000000010010 => 12  [0.99998]
000000010010 => 12
    Time Stamp: 1061
    Average Reward: 1000.0
    Error: 0.0
```

```
    Fitness: 0.999977699477809
    Experience: 48
    Action Set Size: 1.0
    Numerosity: 1
000000111101 => 12  [0.99987]
000000111101 => 12
    Time Stamp: 763
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998670785296495
    Experience: 40
    Action Set Size: 1.0
    Numerosity: 1
000000011010 => 12  [0.99979]
000000011010 => 12
    Time Stamp: 1287
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997923102025773
    Experience: 38
    Action Set Size: 1.0
    Numerosity: 1
000000000100 => 12  [0.99989]
000000000100 => 12
    Time Stamp: 469
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000011100 => 12  [0.99876]
000000011100 => 12
    Time Stamp: 667
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9987620723401149
    Experience: 30
    Action Set Size: 1.0
    Numerosity: 1
000000000001 => 17  [0.99974]
000000000001 => 17
    Time Stamp: 310
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997403877532216
    Experience: 37
    Action Set Size: 1.0
    Numerosity: 1
000000010110 => 12  [0.93128]
000000010110 => 12
    Time Stamp: 3544
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9312812104587673
    Experience: 12
    Action Set Size: 1.0
    Numerosity: 1
000000011111 => 12  [0.99698]
000000011111 => 12
    Time Stamp: 686
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9969777156741089
    Experience: 26
    Action Set Size: 1.0
    Numerosity: 1
000000001010 => 12  [0.99876]
000000001010 => 12
    Time Stamp: 1283
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9987620723401149
    Experience: 30
    Action Set Size: 1.0
    Numerosity: 1
000000011000 => 13  [0.99921]
000000011000 => 13
    Time Stamp: 929
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9992077262976735
    Experience: 32
    Action Set Size: 1.0
    Numerosity: 1
000000110011 => 12  [0.99974]
000000110011 => 12
    Time Stamp: 1063
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997403877532216
    Experience: 37
    Action Set Size: 1.0
    Numerosity: 1
000000000111 => 12  [0.99807]
000000000111 => 12
    Time Stamp: 748
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9980657380314296
    Experience: 28
    Action Set Size: 1.0
    Numerosity: 1
000000001110 => 12  [0.99997]
000000001110 => 12
    Time Stamp: 361
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999721243472612
    Experience: 47
    Action Set Size: 1.0
    Numerosity: 1
000000100011 => 12  [0.99979]
000000100011 => 12
    Time Stamp: 528
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997923102025773
    Experience: 38
    Action Set Size: 1.0
    Numerosity: 1
000000101110 => 12  [0.99949]
```

```
000000101110 => 12
    Time Stamp: 544
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999492944830511
    Experience: 34
    Action Set Size: 1.0
    Numerosity: 1
000000111010 => 12  [0.99876]
```

```
000000111010 => 12
    Time Stamp: 914
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9987620723401149
    Experience: 30
    Action Set Size: 1.0
    Numerosity: 1
```

A.2.4 XCS 5k

```
##########1# => 12  [0.61738]
##########1# => 12
    Time Stamp: 4985
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.6173824060660333
    Experience: 1260
    Action Set Size: 127.68913654760868
    Numerosity: 70
#########1## => 12  [0.60019]
#########1## => 12
    Time Stamp: 5001
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.6001921597819547
    Experience: 1015
    Action Set Size: 131.45905391641577
    Numerosity: 75
#######1#00# => 13  [0.55675]
#######1#00# => 13
    Time Stamp: 4970
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.5567520506490294
    Experience: 119
    Action Set Size: 79.11231905674819
    Numerosity: 36
```

A.2.5 XCS 10k - no GA

```
000000101000 => 11  [0.91410]
000000101000 => 11
    Time Stamp: 8473
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
000000010111 => 12  [0.93128]
000000010111 => 12
    Time Stamp: 8478
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9312812104587673
    Experience: 12
    Action Set Size: 1.0
    Numerosity: 1
000000100101 => 12  [0.99959]
000000100101 => 12
    Time Stamp: 6392
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9995943558644088
    Experience: 35
    Action Set Size: 1.0
    Numerosity: 1
```

```
000000001101 => 12  [0.99989]
000000001101 => 12
    Time Stamp: 4425
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000010100 => 12  [1.00000]
000000010100 => 12
    Time Stamp: 774
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999460806459
    Experience: 75
    Action Set Size: 1.0
    Numerosity: 1
000000100110 => 12  [0.99262]
000000100110 => 12
    Time Stamp: 6871
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9926213761574924
    Experience: 22
    Action Set Size: 1.0
    Numerosity: 1
```

```
000000011110 => 12  [1.00000]
000000011110 => 12
    Time Stamp: 1711
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999998683609518
    Experience: 71
    Action Set Size: 1.0
    Numerosity: 1
000000011100 => 12  [1.00000]
000000011100 => 12
    Time Stamp: 994
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999996786156051
    Experience: 67
    Action Set Size: 1.0
    Numerosity: 1
000000010101 => 12  [0.97748]
000000010101 => 12
    Time Stamp: 8296
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000001000 => 17  [0.99999]
000000001000 => 17
    Time Stamp: 3683
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999908657061105
    Experience: 52
    Action Set Size: 1.0
    Numerosity: 1
000000000010 => 12  [0.98847]
000000000010 => 12
    Time Stamp: 8249
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9884709002460819
    Experience: 20
    Action Set Size: 1.0
    Numerosity: 1
000000101010 => 12  [0.96482]
000000101010 => 12
    Time Stamp: 7197
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9648159797548889
    Experience: 15
    Action Set Size: 1.0
    Numerosity: 1
000000001111 => 12  [0.98847]
000000001111 => 12
    Time Stamp: 8337
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9884709002460819
    Experience: 20
    Action Set Size: 1.0
    Numerosity: 1
000000101100 => 12  [0.99999]
000000101100 => 12
    Time Stamp: 3602
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999941540519107
    Experience: 54
    Action Set Size: 1.0
    Numerosity: 1
000000000111 => 12  [0.99528]
000000000111 => 12
    Time Stamp: 7751
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9952776807407951
    Experience: 24
    Action Set Size: 1.0
    Numerosity: 1
000000011111 => 12  [0.99995]
000000011111 => 12
    Time Stamp: 5475
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999455553657445
    Experience: 44
    Action Set Size: 1.0
    Numerosity: 1
000000111111 => 12  [1.00000]
000000111111 => 12
    Time Stamp: 640
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999995982695064
    Experience: 66
    Action Set Size: 1.0
    Numerosity: 1
000000000001 => 17  [1.00000]
000000000001 => 17
    Time Stamp: 3609
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999962585932229
    Experience: 56
    Action Set Size: 1.0
    Numerosity: 1
000000010000 => 13  [0.99949]
000000010000 => 13
    Time Stamp: 6002
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999492944830511
    Experience: 34
    Action Set Size: 1.0
    Numerosity: 1
000000001110 => 12  [0.91410]
000000001110 => 12
    Time Stamp: 8731
    Average Reward: 1000.0
    Error: 0.0
```

```
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
000000111010 => 12  [1.00000]
000000111010 => 12
    Time Stamp: 523
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999996817163
    Experience: 98
    Action Set Size: 1.0
    Numerosity: 1
000000110001 => 13  [0.99262]
000000110001 => 13
    Time Stamp: 7825
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9926213761574924
    Experience: 22
    Action Set Size: 1.0
    Numerosity: 1
000000110000 => 13  [1.00000]
000000110000 => 13
    Time Stamp: 3534
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999953232415286
    Experience: 55
    Action Set Size: 1.0
    Numerosity: 1
000000000101 => 12  [0.98199]
000000000101 => 12
    Time Stamp: 7631
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.981985781634503
    Experience: 18
    Action Set Size: 1.0
    Numerosity: 1
000000000110 => 12  [0.97185]
000000000110 => 12
    Time Stamp: 7230
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.971852783803911
    Experience: 16
    Action Set Size: 1.0
    Numerosity: 1
000000110101 => 12  [1.00000]
000000110101 => 12
    Time Stamp: 505
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999990286753
    Experience: 93
    Action Set Size: 1.0
    Numerosity: 1
000000011000 => 13  [1.00000]
000000011000 => 13
    Time Stamp: 82
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999962946903
    Experience: 87
    Action Set Size: 1.0
    Numerosity: 1
000000111101 => 12  [1.00000]
000000111101 => 12
    Time Stamp: 681
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999779146326
    Experience: 79
    Action Set Size: 1.0
    Numerosity: 1
000000001010 => 12  [1.00000]
000000001010 => 12
    Time Stamp: 89
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999858653649
    Experience: 81
    Action Set Size: 1.0
    Numerosity: 1
000000111000 => 13  [1.00000]
000000111000 => 13
    Time Stamp: 93
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999998946887614
    Experience: 72
    Action Set Size: 1.0
    Numerosity: 1
000000011001 => 13  [0.91410]
000000011001 => 13
    Time Stamp: 8364
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
000000001001 => 17  [0.93128]
000000001001 => 17
    Time Stamp: 9238
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9312812104587673
    Experience: 12
    Action Set Size: 1.0
    Numerosity: 1
000000101111 => 12  [0.99999]
000000101111 => 12
    Time Stamp: 4789
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999941540519107
    Experience: 54
    Action Set Size: 1.0
    Numerosity: 1
000000111110 => 12  [0.97748]
```

```
000000111110 => 12
    Time Stamp: 7566
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000001011 => 12  [0.99974]
000000001011 => 12
    Time Stamp: 5018
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997403877532216
    Experience: 37
    Action Set Size: 1.0
    Numerosity: 1
000000101011 => 12  [1.00000]
000000101011 => 12
    Time Stamp: 674
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999993783522
    Experience: 95
    Action Set Size: 1.0
    Numerosity: 1
000000010001 => 13  [0.99989]
000000010001 => 13
    Time Stamp: 4847
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000110011 => 12  [0.99528]
000000110011 => 12
    Time Stamp: 7974
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9952776807407951
    Experience: 24
    Action Set Size: 1.0
    Numerosity: 1
000000111100 => 12  [1.00000]
000000111100 => 12
    Time Stamp: 524
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999654916134
    Experience: 77
    Action Set Size: 1.0
    Numerosity: 1
000000110100 => 12  [1.00000]
000000110100 => 12
    Time Stamp: 345
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999909538335
    Experience: 83
    Action Set Size: 1.0
    Numerosity: 1
000000101110 => 12  [0.97748]
000000101110 => 12
    Time Stamp: 7480
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000100000 => 11  [0.91410]
000000100000 => 11
    Time Stamp: 8986
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
000000011010 => 12  [0.99078]
000000011010 => 12
    Time Stamp: 7776
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9907767201968655
    Experience: 21
    Action Set Size: 1.0
    Numerosity: 1
```

A.2.6 XCS 10k

```
#########1## => 12  [0.67954]
#########1## => 12
    Time Stamp: 9984
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.6795355801041069
    Experience: 2627
    Action Set Size: 157.08495787286998
    Numerosity: 94
##########1# => 12  [0.72931]
##########1# => 12
    Time Stamp: 9989
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.7293095749470476
    Experience: 2514
    Action Set Size: 156.91944101706366
    Numerosity: 104
#######1#00# => 13  [0.98840]
#######1#00# => 13
    Time Stamp: 9975
    Average Reward: 1000.0
```

```
    Error: 0.0
    Fitness: 0.9883957624558888
    Experience: 493
    Action Set Size: 120.36751773418345
    Numerosity: 113
######00#00# => 17  [0.90815]
######00#00# => 17
    Time Stamp: 9974
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9081477562761155
    Experience: 271
    Action Set Size: 101.33411866681419
    Numerosity: 83
######10#00# => 11  [0.86133]
######10#00# => 11
    Time Stamp: 9961
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.8613254202089623
    Experience: 240
    Action Set Size: 96.82525289214956
    Numerosity: 82
```

A.2.7 XCS 25k - no GA

```
000000001101 => 12  [1.00000]
000000001101 => 12
    Time Stamp: 17959
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999953232415286
    Experience: 55
    Action Set Size: 1.0
    Numerosity: 1
000000111010 => 12  [0.98559]
000000111010 => 12
    Time Stamp: 22507
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9855886253076024
    Experience: 19
    Action Set Size: 1.0
    Numerosity: 1
000000110000 => 13  [0.97748]
000000110000 => 13
    Time Stamp: 23157
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000011101 => 12  [0.99998]
000000011101 => 12
    Time Stamp: 20666
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999977699477809
    Experience: 48
    Action Set Size: 1.0
    Numerosity: 1
000000010111 => 12  [0.97185]
000000010111 => 12
    Time Stamp: 21935
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.971852783803911
    Experience: 16
    Action Set Size: 1.0
    Numerosity: 1
000000101111 => 12  [0.99999]
000000101111 => 12
    Time Stamp: 19998
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999885821326382
    Experience: 51
    Action Set Size: 1.0
    Numerosity: 1
000000011000 => 13  [0.99959]
000000011000 => 13
    Time Stamp: 20877
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9995943558644088
    Experience: 35
    Action Set Size: 1.0
    Numerosity: 1
000000001001 => 17  [0.99979]
000000001001 => 17
    Time Stamp: 20087
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997923102025773
    Experience: 38
    Action Set Size: 1.0
    Numerosity: 1
000000101010 => 12  [0.91410]
000000101010 => 12
    Time Stamp: 24071
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
000000000010 => 12  [0.97185]
000000000010 => 12
    Time Stamp: 22927
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.971852783803911
    Experience: 16
```

```
    Action Set Size: 1.0
    Numerosity: 1
000000000101 => 12  [0.95602]
000000000101 => 12
    Time Stamp: 23069
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9560199746936111
    Experience: 14
    Action Set Size: 1.0
    Numerosity: 1
000000110100 => 12  [0.97748]
000000110100 => 12
    Time Stamp: 22937
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000111111 => 12  [0.99921]
000000111111 => 12
    Time Stamp: 20197
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9992077262976735
    Experience: 32
    Action Set Size: 1.0
    Numerosity: 1
000000100111 => 12  [0.93128]
000000100111 => 12
    Time Stamp: 23378
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9312812104587673
    Experience: 12
    Action Set Size: 1.0
    Numerosity: 1
000000001111 => 12  [1.00000]
000000001111 => 12
    Time Stamp: 15840
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999779146326
    Experience: 79
    Action Set Size: 1.0
    Numerosity: 1
000000100110 => 12  [0.99989]
000000100110 => 12
    Time Stamp: 21084
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998936628237196
    Experience: 41
    Action Set Size: 1.0
    Numerosity: 1
000000101110 => 12  [1.00000]
000000101110 => 12
    Time Stamp: 15161
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999998683609518
    Experience: 71
    Action Set Size: 1.0
    Numerosity: 1
000000101100 => 12  [0.97748]
000000101100 => 12
    Time Stamp: 22634
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000010100 => 12  [0.89263]
000000010100 => 12
    Time Stamp: 23854
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.892626891341824
    Experience: 10
    Action Set Size: 1.0
    Numerosity: 1
000000111101 => 12  [0.97185]
000000111101 => 12
    Time Stamp: 22988
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.971852783803911
    Experience: 16
    Action Set Size: 1.0
    Numerosity: 1
000000011010 => 12  [0.99983]
000000011010 => 12
    Time Stamp: 20527
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9998338481620619
    Experience: 39
    Action Set Size: 1.0
    Numerosity: 1
000000010001 => 13  [1.00000]
000000010001 => 13
    Time Stamp: 19251
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999976054996627
    Experience: 58
    Action Set Size: 1.0
    Numerosity: 1
000000011011 => 12  [0.95602]
000000011011 => 12
    Time Stamp: 23471
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9560199746936111
    Experience: 14
    Action Set Size: 1.0
    Numerosity: 1
000000101101 => 12  [1.00000]
000000101101 => 12
    Time Stamp: 16790
```

```
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999995982695064
    Experience: 66
    Action Set Size: 1.0
    Numerosity: 1
000000011100 => 12  [1.00000]
000000011100 => 12
    Time Stamp: 14715
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999999332511
    Experience: 105
    Action Set Size: 1.0
    Numerosity: 1
000000111001 => 13  [0.93128]
000000111001 => 13
    Time Stamp: 23811
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9312812104587673
    Experience: 12
    Action Set Size: 1.0
    Numerosity: 1
000000001011 => 12  [1.00000]
000000001011 => 12
    Time Stamp: 15284
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999909538335
    Experience: 83
    Action Set Size: 1.0
    Numerosity: 1
000000110101 => 12  [0.98199]
000000110101 => 12
    Time Stamp: 22820
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.981985781634503
    Experience: 18
    Action Set Size: 1.0
    Numerosity: 1
000000000000 => 17  [0.96482]
000000000000 => 17
    Time Stamp: 22422
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9648159797548889
    Experience: 15
    Action Set Size: 1.0
    Numerosity: 1
000000100001 => 11  [1.00000]
000000100001 => 11
    Time Stamp: 17957
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999995982695064
    Experience: 66
    Action Set Size: 1.0
    Numerosity: 1
000000111110 => 12  [0.96482]
000000111110 => 12
    Time Stamp: 22829
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9648159797548889
    Experience: 15
    Action Set Size: 1.0
    Numerosity: 1
000000110010 => 12  [1.00000]
000000110010 => 12
    Time Stamp: 17327
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999997428924841
    Experience: 68
    Action Set Size: 1.0
    Numerosity: 1
000000000001 => 17  [0.99974]
000000000001 => 17
    Time Stamp: 20280
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997403877532216
    Experience: 37
    Action Set Size: 1.0
    Numerosity: 1
000000110011 => 12  [0.91410]
000000110011 => 12
    Time Stamp: 23838
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9141015130734592
    Experience: 11
    Action Set Size: 1.0
    Numerosity: 1
000000110001 => 13  [0.99876]
000000110001 => 13
    Time Stamp: 21760
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9987620723401149
    Experience: 30
    Action Set Size: 1.0
    Numerosity: 1
000000111000 => 13  [0.99921]
000000111000 => 13
    Time Stamp: 20615
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9992077262976735
    Experience: 32
    Action Set Size: 1.0
    Numerosity: 1
000000000100 => 12  [1.00000]
000000000100 => 12
    Time Stamp: 17297
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999987740158274
    Experience: 61
    Action Set Size: 1.0
```

```
    Numerosity: 1
000000011111 => 12  [0.99979]
000000011111 => 12
    Time Stamp: 20632
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997923102025773
    Experience: 38
    Action Set Size: 1.0
    Numerosity: 1
000000111100 => 12  [0.93128]
000000111100 => 12
    Time Stamp: 23758
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9312812104587673
    Experience: 12
    Action Set Size: 1.0
    Numerosity: 1
000000010000 => 13  [0.97748]
000000010000 => 13
    Time Stamp: 22692
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000010110 => 12  [0.89263]
000000010110 => 12
    Time Stamp: 23699
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.892626891341824
    Experience: 10
    Action Set Size: 1.0
    Numerosity: 1
000000000011 => 12  [0.99845]
000000000011 => 12
    Time Stamp: 20767
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9984525904251437
    Experience: 29
    Action Set Size: 1.0
    Numerosity: 1
000000110110 => 12  [0.99758]
000000110110 => 12
    Time Stamp: 21405
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.997582172539287
    Experience: 27
    Action Set Size: 1.0
    Numerosity: 1
```

A.2.8 XCS 25k

```
#########1## => 12  [0.83936]
#########1## => 12
    Time Stamp: 24990
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.8393557043818822
    Experience: 6644
    Action Set Size: 174.5116942841693
    Numerosity: 131
##########1# => 12  [0.78616]
##########1# => 12
    Time Stamp: 24979
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.786163298369161
    Experience: 6867
    Action Set Size: 186.3288624137215
    Numerosity: 127
#######1#00# => 13  [0.99293]
#######1#00# => 13
    Time Stamp: 24988
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.992929511086371
    Experience: 1632
    Action Set Size: 140.19315361135267
    Numerosity: 134
######10#00# => 11  [0.95084]
######10#00# => 11
    Time Stamp: 24993
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9508432572598198
    Experience: 763
    Action Set Size: 119.46096408567944
    Numerosity: 109
######00#00# => 17  [0.99405]
######00#00# => 17
    Time Stamp: 24950
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9940463286133433
    Experience: 788
    Action Set Size: 110.9822021406458
    Numerosity: 106
```

A.2.9 XCS 50k - no GA

```
000000010110 => 12  [0.99807]
000000010110 => 12
    Time Stamp: 46711
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9980657380314296
    Experience: 28
    Action Set Size: 1.0
    Numerosity: 1
000000111011 => 12  [0.97748]
000000111011 => 12
    Time Stamp: 48034
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9774822270431288
    Experience: 17
    Action Set Size: 1.0
    Numerosity: 1
000000100010 => 12  [1.00000]
000000100010 => 12
    Time Stamp: 36416
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999999658244
    Experience: 108
    Action Set Size: 1.0
    Numerosity: 1
000000101000 => 11  [0.99979]
000000101000 => 11
    Time Stamp: 46474
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997923102025773
    Experience: 38
    Action Set Size: 1.0
    Numerosity: 1
000000011001 => 13  [0.99968]
000000011001 => 13
    Time Stamp: 45725
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.999675484691527
    Experience: 36
    Action Set Size: 1.0
    Numerosity: 1
000000110010 => 12  [0.89263]
000000110010 => 12
    Time Stamp: 49058
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.892626891341824
    Experience: 10
    Action Set Size: 1.0
    Numerosity: 1
000000001011 => 12  [0.99807]
000000001011 => 12
    Time Stamp: 45824
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9980657380314296
    Experience: 28
    Action Set Size: 1.0
    Numerosity: 1
000000010100 => 12  [0.99959]
000000010100 => 12
    Time Stamp: 46861
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9995943558644088
    Experience: 35
    Action Set Size: 1.0
    Numerosity: 1
000000010011 => 12  [0.99622]
000000010011 => 12
    Time Stamp: 47057
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9962221445926361
    Experience: 25
    Action Set Size: 1.0
    Numerosity: 1
000000001001 => 17  [0.99997]
000000001001 => 17
    Time Stamp: 44456
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999651554340765
    Experience: 46
    Action Set Size: 1.0
    Numerosity: 1
000000110111 => 12  [0.89263]
000000110111 => 12
    Time Stamp: 48699
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.892626891341824
    Experience: 10
    Action Set Size: 1.0
    Numerosity: 1
000000010111 => 12  [1.00000]
000000010111 => 12
    Time Stamp: 37847
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999999726595
    Experience: 109
    Action Set Size: 1.0
    Numerosity: 1
000000100110 => 12  [0.97185]
000000100110 => 12
    Time Stamp: 48849
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.971852783803911
    Experience: 16
    Action Set Size: 1.0
    Numerosity: 1
000000011011 => 12  [1.00000]
000000011011 => 12
    Time Stamp: 40466
    Average Reward: 1000.0
    Error: 0.0
```

```
    Fitness: 0.9999999927630668
    Experience: 84
    Action Set Size: 1.0
    Numerosity: 1
000000101010 => 12  [0.99758]
000000101010 => 12
    Time Stamp: 46579
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.997582172539287
    Experience: 27
    Action Set Size: 1.0
    Numerosity: 1
000000100011 => 12  [0.97185]
000000100011 => 12
    Time Stamp: 47672
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.971852783803911
    Experience: 16
    Action Set Size: 1.0
    Numerosity: 1
000000111101 => 12  [0.97185]
000000111101 => 12
    Time Stamp: 48881
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.971852783803911
    Experience: 16
    Action Set Size: 1.0
    Numerosity: 1
000000001110 => 12  [0.99262]
000000001110 => 12
    Time Stamp: 46962
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9926213761574924
    Experience: 22
    Action Set Size: 1.0
    Numerosity: 1
000000110101 => 12  [0.93128]
000000110101 => 12
    Time Stamp: 48621
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9312812104587673
    Experience: 12
    Action Set Size: 1.0
    Numerosity: 1
000000100100 => 12  [0.98199]
000000100100 => 12
    Time Stamp: 48188
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.981985781634503
    Experience: 18
    Action Set Size: 1.0
    Numerosity: 1
000000000001 => 17  [0.99993]
000000000001 => 17
    Time Stamp: 45648
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999319442071806
    Experience: 43
    Action Set Size: 1.0
    Numerosity: 1
000000000000 => 17  [0.94502]
000000000000 => 17
    Time Stamp: 48427
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9450249683670139
    Experience: 13
    Action Set Size: 1.0
    Numerosity: 1
000000100111 => 12  [0.98199]
000000100111 => 12
    Time Stamp: 46902
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.981985781634503
    Experience: 18
    Action Set Size: 1.0
    Numerosity: 1
000000100001 => 11  [0.99807]
000000100001 => 11
    Time Stamp: 46199
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9980657380314296
    Experience: 28
    Action Set Size: 1.0
    Numerosity: 1
000000000110 => 12  [0.97185]
000000000110 => 12
    Time Stamp: 48993
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.971852783803911
    Experience: 16
    Action Set Size: 1.0
    Numerosity: 1
000000011100 => 12  [0.98847]
000000011100 => 12
    Time Stamp: 46063
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9884709002460819
    Experience: 20
    Action Set Size: 1.0
    Numerosity: 1
000000110110 => 12  [0.99262]
000000110110 => 12
    Time Stamp: 47506
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9926213761574924
    Experience: 22
    Action Set Size: 1.0
    Numerosity: 1
000000101001 => 11  [0.99979]
```

```
000000101001 => 11
    Time Stamp: 44069
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9997923102025773
    Experience: 38
    Action Set Size: 1.0
    Numerosity: 1
000000101100 => 12  [0.99999]
000000101100 => 12
    Time Stamp: 44210
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999885821326382
    Experience: 51
    Action Set Size: 1.0
    Numerosity: 1
000000111000 => 13  [0.99758]
000000111000 => 13
    Time Stamp: 47020
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.997582172539287
    Experience: 27
    Action Set Size: 1.0
    Numerosity: 1
000000011010 => 12  [1.00000]
000000011010 => 12
    Time Stamp: 36623
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999999999942663
    Experience: 116
    Action Set Size: 1.0
    Numerosity: 1
000000011110 => 12  [1.00000]
000000011110 => 12
    Time Stamp: 43872
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999984675197842
    Experience: 60
    Action Set Size: 1.0
    Numerosity: 1
000000011111 => 12  [0.94502]
000000011111 => 12
    Time Stamp: 48276
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9450249683670139
    Experience: 13
    Action Set Size: 1.0
    Numerosity: 1
000000100101 => 12  [0.99991]
000000100101 => 12
    Time Stamp: 46100
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999149302589757
    Experience: 42
    Action Set Size: 1.0
    Numerosity: 1
000000010000 => 13  [0.99758]
000000010000 => 13
    Time Stamp: 47013
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.997582172539287
    Experience: 27
    Action Set Size: 1.0
    Numerosity: 1
000000111110 => 12  [1.00000]
000000111110 => 12
    Time Stamp: 40946
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999998946887614
    Experience: 72
    Action Set Size: 1.0
    Numerosity: 1
000000101111 => 12  [0.99997]
000000101111 => 12
    Time Stamp: 43190
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999651554340765
    Experience: 46
    Action Set Size: 1.0
    Numerosity: 1
000000000101 => 12  [1.00000]
000000000101 => 12
    Time Stamp: 41050
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9999998946887614
    Experience: 72
    Action Set Size: 1.0
    Numerosity: 1
000000010001 => 13  [0.95602]
000000010001 => 13
    Time Stamp: 47982
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9560199746936111
    Experience: 14
    Action Set Size: 1.0
    Numerosity: 1
```

A.2.10 XCS 50k

```
#########1## => 12  [0.74758]
#########1## => 12
    Time Stamp: 49983
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.7475818834062665
    Experience: 13496
    Action Set Size: 176.9677180244134
    Numerosity: 116
##########1# => 12  [0.70552]
##########1# => 12
    Time Stamp: 49983
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.7055237635160592
    Experience: 13624
    Action Set Size: 197.44407451716674
    Numerosity: 126
#######1#00# => 13  [0.98853]
#######1#00# => 13
    Time Stamp: 49999
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9885283302692095
    Experience: 3354
    Action Set Size: 141.80496111657925
    Numerosity: 135
######00#00# => 17  [0.98375]
######00#00# => 17
    Time Stamp: 49974
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.9837493831868861
    Experience: 1547
    Action Set Size: 112.64795548818489
    Numerosity: 106
######10#00# => 11  [0.99421]
######10#00# => 11
    Time Stamp: 49952
    Average Reward: 1000.0
    Error: 0.0
    Fitness: 0.994205954497984
    Experience: 1467
    Action Set Size: 109.04690303581262
    Numerosity: 98
```

A.3 Training summaries

A.3.1 XCS training summary - DSC

XCS configuration	Training time (s)	Average fitness	Number of learned situations
XCS 1k - no GA	11	0.92781	5
XCS 1k	12	0.57562	1
XCS 5k - no GA	13	0.99	42
XCS 5k	15	0.73143	4
XCS 10k - no GA	35	0.99	41
XCS 10k	39	0.68223	4
XCS 25k - no GA	50	0.99	44
XCS 25k	51	0.670	6
XCS 50k - no GA	78	0.99	51
XCS 50k	81	0.715584	5
XCS 100k	237	0.72835	6

Table A.1: Summary of XCS training performance for the DSC agent, listing the training time, fitness, number of learned situations and fitness/situation performance

A.3.2 XCS training summary - DSP

XCS configuration	Training time (s)	Average fitness	Number of learned situations
XCS 1k - no GA	-	-	-
XCS 1k	-	-	-
XCS 5k - no GA	11	0.99	47
XCS 5k	14	0.59144	3
XCS 10k - no GA	44	0.99	43
XCS 10k	45	0.833346	5
XCS 25k - no GA	52	0.98	43
XCS 25k	57	0.912648	5
XCS 50k - no GA	72	0.992	39
XCS 50k	87	0.883918	5
XCS 100k	240	0.911158	5

Table A.2: Summary of XCS training performance for the DSP agent, listing the training time, fitness, number of learned situations and fitness/situation performance

Action identifier	Action description
3	Change response type
4	Switch ontology
6	Adapt protocol type
8	Decrease request specificy
10	Decrease request rate
11	Increase response precision
13	Increase response rate
17	Do nothing

Table A.3: Action identifiers and description for the I-IOP agent in the IoT ecosystem simulator.

A.4 Configurations

A.4.1 Simulator configuration

Configuration parameter	Description
XCS config	The used XCS configuration by the agent
Agent active	Where the agent is operational or not. This is used to evaluate the non-agent vs. the agent based deployments
Provider utility	The utility function employed by the provider. By adjusting this parameter, the agent is trained to optimize for runtime interoperability in a different direction.
Number of providers	Number of provider systems to deploy in simulation
Mapping parameters	The semantic mapping starting parameters which influence how ontologies are mapped
Provider heterogeneity	Determines whether provider systems are different
Ontologies	The ontologies to be used by the systems. In case of homogeneous ontologies, this will only be one to be used by all system in the simulation in which case semantic interoperability will always be 1.
Buffer	Simulated communication buffer

Table A.4: Configuration space of the IoT ecosystem simulator.

Configuration space

A.4.1.1 Scenario S_1

Configuration parameter	Value
Agent_active	[True,True,False]
Provider - API config	[0,1]
Provider - ontologies	[0,0]
Provider - number of providers	2
Provider - utility	utilities.utility_max_efficiency
Provider - prob. unavailability	[0,0]
Provider - prob. firmware	[0,0]
Provider - prob. delay	[0,0]
Provider - prob. com failure	[0,0]
Consumer - request_config	0
Consumer - ontology	0
Consumer - utility	utilities.utility_max_efficiency
Consumer - prob. com failure	0
Required job types	[data,actuation]
Request specificy	1
iterations	20

Table A.5: Simulator configuration of scenario S_1

A.4.1.2 Scenario S_2

Configuration parameter	Value
Agent_active	[True,True,False]
Provider - API config	[0,1,2]
Provider - ontologies	[0,0,0]
Provider - number of providers	3
Provider - utility	utilities.utility_max_efficiency
Provider - prob. unavailability	[0,0]
Provider - prob. firmware	[0,0]
Provider - prob. delay	[0,0]
Provider - prob. com failure	[0,0]
Consumer - request_config	0
Consumer - ontology	0
Consumer - utility	utilities.utility_max_efficiency
Consumer - prob. com failure	0
Required job types	[data,actuation]
Request specificy	1
iterations	20

Table A.6: Simulator configuration of scenario S_2

A.4.1.3 Scenario S_3

Configuration parameter	Value
Agent_active	[True,True,False]
Provider - API config	[0,1,3]
Provider - ontologies	[0,0,0]
Provider - number of providers	3
Provider - utility	utilities.utility_max_efficiency
Provider - prob. unavailability	[0,0]
Provider - prob. firmware	[0,0]
Provider - prob. delay	[0,0]
Provider - prob. com failure	[0,0]
Consumer - request_config	0
Consumer - ontology	0
Consumer - utility	utilities.utility_max_efficiency
Consumer - prob. com failure	0
Required job types	[data,actuation]
Request specificy	1
iterations	20

Table A.7: Simulator configuration of scenario S_3

A.4.1.4 Scenario S_4

Configuration parameter	Value
Agent_active	[True,True,False]
Provider - API config	[0]
Provider - ontologies	[0]
Provider - number of providers	1
Provider - heterogeneity	1
Provider - utility	utilities.utility_max_efficiency
Provider - prob. unavailability	[0]
Provider - prob. firmware	[0.1]
Provider - prob. delay	[0.05]
Provider - prob. com failure	[0]
Consumer - request_config	0
Consumer - ontology	0
Consumer - utility	utilities.utility_max_efficiency
Consumer - prob. com failure	0
Required job types	[data]
Request specificy	1
iterations	20

Table A.8: Simulator configuration of scenario S_4

A.4.1.5 Scenario S_F

Configuration parameter	Value
Agent_active	[True,True,False]
Provider - API config	[0,1,2]
Provider - ontologies	[1,0,1]
Provider - number of providers	3
Provider - heterogeneity	1
Provider - utility	utilities.utility_max_efficiency
Provider - prob. unavailability	[0,0,0]
Provider - prob. firmware	[0,0]
Provider - prob. delay	[0,0]
Provider - prob. com failure	[0,0]
Consumer - request_config	0
Consumer - ontology	0
Consumer - utility	utilities.utility_max_efficiency
Consumer - prob. com failure	0
Required job types	[data,actuation,actuation_1]
Request specificy	1
iterations	20

Table A.9: Simulator configuration of scenario S_F

A.4.1.6 Scenario S_P

Configuration parameter	Value
Agent_active	[True,True,False]
Provider - API config	[1,2]
Provider - ontologies	[1,0]
Provider - number of providers	2
Provider - heterogeneity	1
Provider - utility	utilities.utility_max_efficiency
Provider - prob. unavailability	[0,0]
Provider - prob. firmware	[0,0.4]
Provider - prob. delay	[0,0]
Provider - prob. com failure	[0,0]
Consumer - request_config	0
Consumer - Ontology heterogeneity	0
Consumer - utility	utilities.utility_max_efficiency_sp
Consumer - prob. com failure	0
Required job types	[data,actuation]
Request specificy	1
iterations	20

Table A.10: Simulator configuration of scenario S_P

A.4.2 XCS configurations

A.4.2.1 XCS configuration space

XCS-Parameter	default value
Population size	2000
Learning rate	0.15
Discount rate	0.71
Exploration probability	0.01
Accuracy coefficient	0.1
error threshold	0.01
GA threshold	1
Deletion consideration threshold	1
Minimum number of actions threshold	0
Subsumption consideration threshold	20
Wildcard probability	0.998
Mutation probability	0.004
Crossover probability	0.8
Initial value for the prediction	$1\ e^{-05}$
Initial value for the error	$1\ e^{-05}$
Initial value for the fitness	$1\ e^{-05}$
Fitness reduction factor	0.1

Table A.11: The XCS configuration space

A.4.2.2 XCS 50k

XCS-Parameter	default value
Population size	2000
Learning rate	0.2
Discount rate	0.0
Exploration probability	0.5
Accuracy coefficient	0.1
error threshold	1
GA threshold	25
Deletion consideration threshold	1
Minimum number of actions threshold	0
Subsumption consideration threshold	20
Wildcard probability	0.5
Mutation probability	0.004
Crossover probability	0.5
Initial value for the prediction	$1\ e^{-05}$
Initial value for the error	$1\ e^{-05}$
Initial value for the fitness	$1\ e^{-05}$
Fitness reduction factor	0.1

Table A.12: The configuration for the XCS agent with 50.000 trained iterations

A.4.2.3 XCS 100k

The same configuration as in XCS50k was used, except that the agent was trained for 100.000 iterations.

A.4.2.4 XCS 5k - no GA

XCS-Parameter	default value
Population size	2000
Learning rate	0.2
Discount rate	0.0
Exploration probability	0.0
Accuracy coefficient	0.1
error threshold	0.1
GA threshold	1000000
Deletion consideration threshold	1
Minimum number of actions threshold	0
Subsumption consideration threshold	20
Wildcard probability	0.0
Mutation probability	0.000
Crossover probability	0
Initial value for the prediction	$1\ e^{-05}$
Initial value for the error	$1\ e^{-05}$
Initial value for the fitness	$1\ e^{-05}$
Fitness reduction factor	0.1

Table A.13: The configuration for the XCS agent with 5.000 trained iterations an inactive GA

Bibliography

[ABI+11] C. H. Asuncion, C. Boldyreff, S. Islam, M. Leonard, and B. Thalheim. Pragmatic interoperability in the enterprise - a research agenda. *Theoretical Computer Science - TCS*, 731:0, 01 2011.

[Ack71] R. L. Ackoff. *Towards a System of Systems Concepts*, volume 17. INFORMS, 1971.

[aH15] Arbeitskreis Smart Service Welt / acatech (Hrsg). Smart Service Welt - Umsetzungsempfehlungen fuer das Zukunftsprojekt Internet-basierte Dienste fuer die Wirtschaft. *Deutsche Akademie der Technikwissenschaften*, 2015.

[AIM10] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[AO16] D. B. Abeywickrama and E. Ovaska. A survey of autonomic computing methods in digital service ecosystems. *Service Oriented Computing and Applications*, pages 1–31, 2016.

[Asu10] C. Asuncion. *Pragmatic Interoperability: A Systematic Review of Published Definitions*, volume 326, pages 164–175. Springer Berlin, 08 2010.

[AV11] C. H. Asuncion and M. Van Sinderen. Towards pragmatic interoperability in the New Enterprise - A survey of approaches. *Lecture Notes in Business Information Processing*, 76 LNBIP:132–145, 2011.

[Bar11] B. Bara. Cognitive pragmatics the mental processes of communication. *Intercultural Pragmatics*, 8, 09 2011.

[Bat13] T. Batista. *Middleware Solutions for the Internet of Things*. Springer, 09 2013.

[BD15] S. Billaud and N. Daclin. Interoperability as a Key Concept for the Control and Evolution of the System of Systems (SoS). *International Federation for Information Processing*, 76(March):118–131, 2015.

[BHKL13] A. Ben Hamida, F. Kon, and N. et al. Lago. Integrated choreos middleware - enabling large-scale, qos-aware adaptive choreographies. 09 2013.

[BI15] A. Bennaceur and V. Issarny. Automated synthesis of mediators to support component interoperability. *IEEE Transactions on Software Engineering*, 41(3):221–240, 2015.

[BL15] M. Blackstock and R. Lea. Iot interoperability: A hub-based approach. *2014 International Conference on the Internet of Things, IOT 2014*, pages 79–84, 02 2015.

[Bor14] E. Borgia. The Internet of Things vision: Key features, applications and open issues. *Computer Communications*, 54:1–31, 2014.

[BPGG11] G. Blair, M. Paolucci, P. Grace, and N. Georgantas. Interoperability in Complex Distributed Systems Interoperability Barriers : Dimensions of Heterogeneity. *Europe*, pages 1–26, 2011.

[Brö17] A. Bröring. Enabling IoT Ecosystems through Platform Interoperability. *IEEE Software*, 34(1):54–61, 2017.

[BS06] J. Boardman and B. Sauser. The meaning of system of systems. *System of Systems Engineering*, 0, 04 2006.

[BST16] G. Blair, D. Schmidt, and C. Taconet. Middleware for Internet distribution in the context of cloud computing and the Internet of Things: Editorial Introduction. *Annals of Telecommunications*, 71(3-4):87–92, 2016.

[BTW15] K. Bellman, S. Tomforde, and R. Würtz. Interwoven systems: Self-improving systems integration. *Proceedings - 2014 IEEE 8th International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014*, pages 123–127, 03 2015.

[Bul15] L. Bull. A brief history of learning classifier systems: from CS-1 to XCS and its variants. *Evolutionary Intelligence*, 8(2-3):55–70, 2015.

[But15] Martin Butz. Learning classifier systems. In *Springer Handbook of Computational Intelligence*, pages 2367–2388, 07 2015.

[CBBZ18] M. Chernyshev, Z. Baig, O. Bello, and S. Zeadally. Internet of Things (IoT): Research , Simulators and Testbeds. *IEEE Internet of Things Journal*, 5(3):1637–1647, 2018.

[CBF+16] Andrea Ceccarelli, Andrea Bondavalli, Bernhard Froemel, Oliver Hoeftberger, and Hermann Kopetz. *Basic Concepts on Systems of Systems*, pages 1–39. Springer International Publishing, Cham, 2016.

[CD12] V. Chapurlat and N. Daclin. *System interoperability: Definition and proposition of interface model in MBSE context*, volume 14. IFAC Proceedings Volumes, 2012.

[CDE06] D. Chen, M. Dassisti, and B. Elvesaeter. Enterprise interoperability-framework and knowledge corpus., 2006.

[CFMP05] D. Carney, D. Fisher, E. Morris, and P. Place. Some Current Approaches to Interoperability. *Integration The Vlsi Journal*, 1(August):27, 2005.

[Che09] B. et al. Cheng. Software engineering for self-adaptive systems: A research roadmap. In *"Software Engineering for Self-Adaptive Systems"*, volume 5525, pages 1–26. "Springer Berlin Heidelberg", 01 2009.

[CHP18] P. Constantinides, O. Henfridsson, and G. Parker. Platforms and infrastructures in the digital age. *Information Systems Research*, 29, 05 2018.

[CN15] J. Cardoso and S. Nickel. *Fundamentals of Service Systems*, volume 1. Springer, 2015.

[Coo] Gabler wirtschaftslexikon - definition coopetition. https://wirtschaftslexikon.gabler.de/definition/coopetition-27127. Accessed: 2019-12-27.

[CW06] E. Chang and M. West. Digital ecosystems a next generation of the collaborative environment. *The Eight International Conference on Information Systems*, pages 3–23, 2006.

[Del13] J. Delgado. Service Interoperability in the Internet of Things. *Comput. Technol. SCI*, 460:51–87, 2013.

[DM14] N. Daclin and S. Mallek. Capturing and Structuring Interoperability Requirements: A Framework for Interoperability Requirements . *Enterprise Interoperability VI*, pages 41–51, 2014.

[DMA13] A. Didandeh, N. Mirbakhsh, and M. Afsharchi. Concept learning games. *Information Systems Frontiers*, 15(4):653–676, 2013.

[dRSB17] M. de Reuver, C. Sørensen, and R. C. Basole. The digital platform: a research agenda. *Journal of Information Technology*, pages 1–12, 2017.

[DV15] W. Damm and A. Vincentelli. A conceptual model of system of systems. In *SWEC 2015*, pages 19–27, 04 2015.

[ERA10] M. Eisenhauer, P. Rosengren, and P. Antolin. Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems. *The Internet of Things*, pages 367–373, 2010.

[FCGD07] T. C. Ford, J. M. Colombi, S. R. Graham, and Jacques D.R. A Survey on Interoperability Measurement. *Information Systems*, 2007.

[FGCJ08] T. Ford, S. Graham, J. Colombi, and D. Jacques. Measuring System Interoperability (An i-Score Improvement). *Conference on Systems Engineering Research, April 4-5*, (November 2016):1–10, 2008.

[Fis06] D. A Fisher. An Emergent Perspective on Interoperation in Systems of Systems. Technical report, Software Engineering Institute, Carnegie Mellon University, 2006.

[For08] T. Ford. *Interoperability Measurement.* Biblioscholar, 08 2008.

[Gas15] U. Gasser. Interoperability in the Digital Ecosystem. *SSRN Electronic Journal*, 7641:36, 2015.

[GBF+18] J. Guth, Uwe Breitenbücher, M. Falkenthal, P. Fremantle, O. Kopp, F. Leymann, and L. Reinfurt. *A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences*, pages 81–101. 01 2018.

[GC02] A. Gawer and M. A. Cusumano. *Platform Leadership*, volume 1. Harvard Business School Press, Boston, MA, USA, 2002.

[GC13] A. Gawer and M. Cusumano. Industry Platform and Ecosystem Innovation. *Journal of Product Innovation and Management*, 31(3):417–433, 2013.

[GCB+14] M.B. Gonçalves, E. Cavalcante, T. Batista, F. Oquendo, and E.Y. Nakagawa. Towards a conceptual model for software-intensive system-of-systems. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 2014-Janua(January):1605–1610, 2014.

[GIB12] E. Grousset, V. Issarny, and A. et al. Bertolino. Connect; project final report - use and dissemination of foreground, 12 2012.

[Gmb16] Berg Insight GmbH. IoT Platforms and Software. Technical report, Berg Insight, 2016.

[GMLF18] R. Gravina, M. Manso, A. Liotta, and G. Fortino. *Integration, Interconnection, and Interoperability of IoT Systems*, volume 1. Springer, 2018.

[GP12] U. Gasser and J. Palfrey. *Interop: The Promise and Perils of Highly Interconnected Systems*, volume 1. Basic Books, 2012.

[GPP+16] M. Ganzha, M. Paprzycki, W. Pawlowski, P. Szmeja, and K. Wasielewska. Semantic interoperability in the Internet of Things: An overview from the INTER-IoT perspective. *Journal of Network and Computer Applications*, 2016.

[Gué14] W. Guédria. A conceptual framework for enterprise interoperability. *Int. J. E-Bus. Res.*, 10:54–64, July 2014.

[GW12] F. Guo and M. Wang. Quantitative measurement of interoperability by using Petri net. *Journal of Computational Information Systems*, 8(8):3245–3252, 2012.

[GZ04] C.V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.

[HF56] A Hall and R Fagen. Definition of system. In *General systems*, volume 1, pages 18–28, 1956.

[Hoa16] S. Hoare. A study of the state-of-the-art of PaaS interoperability. *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering - EASE '16*, pages 1–4, 2016.

[HS11] Cooperative Hybrid and Objects Sensor. CHOSeN Project Report Extended Publishable Summary. pages 1–42, 2011.

[Ini15] IEEE Internet Initiative. Towards a definition of the Internet of Things (IoT). page 27, 2015.

[IOKP16] A. Immonen, E. Ovaska, J. Kalaoja, and D. Pakkala. A service requirements engineering method for a digital service ecosystem. *Service Oriented Computing and Applications*, 10(2):151–172, 2016.

[IoT] BIG IoT. Big iot eu project. http://big-iot.eu/. Accessed: 2019-12-12.

[IoT16] IoT Analytics GmbH. IoT Platform Market Report 2015-2021. Technical Report January, IoT Analytics, 2016.

[JB12] F-W. Jaekel and G. et al. Benguria. A methodology for interoperability evaluation based on causal performance measurement models. *Enterprise Interoperability*, 5:61 – 70, 2012.

[JWDM13] E. Jones-Wyatt, J.C. Domercant, and D. Mavris. A reliability-based measurement of interoperability for systems of systems. In *SysCon 2013 - 7th Annual IEEE International Systems Conference, Proceedings*, pages 408–413, 04 2013.

[KGR16] M. Kostoska, M. Gusev, and S. Ristov. An Overview of Cloud Interoperability. *Federated conference on computer science and ifnromation systems*, 8:873–876, 2016.

[KH18] Denis Kramer and Joerg Haehner. Beyond semantic interoperability in iot ecosystems. In *IOT '18: Proceedings of the 8th International Conference on the Internet of Things*, pages 1–4, 10 2018.

[KH19] Denis Kramer and Joerg Haehner. A self-aware systems approach for interoperability in iot ecosystems. In *32nd GI/ITG International Conference on Architecture of Computing Systems May 20 - 21, 2019, Technical University of Denmark, Copenhagen, Denmark Workshop Proceedings*, volume 6. VDE, 2019.

[KK17] S. Kounev and J. O. Kephart. *Self-Aware Computing Systems*, volume 1. Springer, 2017.

[KMV+15] C. Krupitzer, F. Maximilian, S. Vansyckel, G. Schiele, and C. Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17:184–206, 2015.

[LBB12] W. Li, Y. Badr, and F. Biennier. Digital Ecosystems: Challenges and Prospects. *Proceedings of the International Conference on Management of Emergent Digital EcoSystems (MEDES '12)*, pages 117–122, 2012.

[LCP+11] P. R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and X. Yao. A survey of self-awareness and its application in computing systems. *Proceedings - 2011 5th IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2011*, pages 102–107, 2011.

[LGP16] G. S. S. Leal, W. Guedria, and H. Panetto. Towards a Comparative Analysis of Interoperability Assessment Approaches for Collaborative Enterprise Systems. *ISPE - International Conference on Transdiciplinary Engineering*, 2016.

[Liu07] K. Liu. Pragmatic Computing - A Semiotic Perspective to Web Services. *CCIS*, 23:3–15, 2007.

[LK91] A. M. Law and D. W. Kelton. *Simulation Modeling & Analysis*, volume 2. University of Michigan, 1991.

[LLL14] S. Liu, W. Li, and K. Liu. Assessing Pragmatic Interoperability of Information Systems from a Semiotic Perspective. *Iciso 2014*, 449:32–41, 2014.

[LPR+16] P. R Lewis, M. Platzner, B. Rinner, J. Tørresen, and X. Yao. *Self-aware Computing Systems: An Engineering Approach*, volume 1. Springer, 2016.

[Luc16] S. Lucero. IoT platforms : enabling the Internet of Things, 2016.

[Luk15] J. Lukkien. A Summary on Systems of Systems Engineering. 2015.

[Luk16] J. Lukkien. A Systems of Systems perspective on the Internet of Things. *ACM SIGBED Review*, 13(June):56–62, 2016.

[LY02] Kalle Lyytinen and Youngjin Yoo. Ubiquitous computing. *Communications of the ACM*, 45(12):63–96, 2002.

[Mai98] M. W. Maier. Architecting Principles for Systems-of-Systems. *Systems Engineering*, 1:267–284, 1998.

[MCG+14] P. Maia, E. Cavalcante, P. Gomes, T. Batista, F. C Delicato, and P. F. Pires. On the Development of Systems-of-Systems based on the Internet of Things : A Systematic Mapping, 2014.

[MDC10] S. Mallek, N. Daclin, and V. Chapurlat. *Towards a Conceptualisation of Interoperability Requirements*, volume 4, pages 439–448. Springer, 01 2010.

[MMP19] F. Marino, C. Moiso, and M. Petracca. Automatic contract negotiation, service discovery and mutual authentication solutions: A survey on the enabling technologies of the forthcoming IoT ecosystems. *Computer Networks*, 148:176–195, 2019.

[MMST16] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma. A gap analysis of Internet-of-Things platforms. *Computer Communications*, 8990:5–16, 2016.

[MSDC12] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10:1497–1516, 2012.

[Muf09] M. Muffatti. *The Impact of SOA on Interoperability: a Systematic Literature Review*. PhD thesis, Politecnico Milano, 2009.

[Mun02] S. Munk. An analysis of basic interoperability related terms, system of interoperability types. *Academic and Applied Research in Military Sciences*, 1(1):117–131, 2002.

[MVEP03] P. Mcburney, R.M. Van Eijk, and S. et al. Parsons. A Dialogue Game Protocol for Agent Purchase Negotiations. *Autonomous Agents and Multi-Agent Systems*, 1:235–273, 2003.

[Nay15] M.M. Nayebpour. The interoperability index model: Improving the I-Score model for interoperability measurement. *International Journal of Advanced Research in Engineering and Applied Sciences*, 4(11):24–36, 2015.

[NDBC16] F. W. Neiva, J. M. N. David, R. Braga, and F. Campos. Towards pragmatic interoperability to support collaboration: A systematic review and mapping of the literature. *Information and Software Technology*, 72:137–150, 2016.

[NGC09] Y. Naudet, W. Guedria, and D. Chen. Systems science for enterprise interoperability. *Proceedings - 2009 International Conference on Interoperability for Enterprise Software and Applications, IESA 2009*, pages 107–113, 2009.

[NHRdR18] R. Nicolescu, M. Huth, P. Radanliev, and D. de Roure. Mapping the values of IoT. *Journal of Information Technology*, pages 1–16, 2018.

[NKK+09] M. Nagy, A. Katasonov, O. Khriyenko, S. Nikitin, M. Szydlowski, and Terzivan. Challenges of middleware for the internet of things. *Automation Control - Theory and Practice*, 2009.

[NLGC10] Y. Naudet, T. Latour, W. Guedria, and D. Chen. Towards a systemic formalisation of interoperability. *Computers in Industry*, 61(2):176–185, 2010.

[OA16] F. A. Oliehoek and C. Amato. *A concise introduction to Decentralized POMDPs*, volume 1. Springer International Publishing, 2016.

[PG03] M. Papazoglou and D. Georgakopoulos. Introduction: Service-oriented computing. *Communications of the ACM*, 46:24–28, 10 2003.

[PL05] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[PT02] D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, Jun 2002.

[PZJGR16] H. Panetto, M. Zdravkovic, R. Jardim-Goncalves, and D. et al. Romero. New perspectives for the future interoperable enterprise systems. *Computers in Industry*, 79:47–63, 2016.

[Raj11] R Rajmohan. A Survey on Problems in Distributed UDDI. *International Journal of Computer Applications*, 36(3):1–7, 2011.

[RBD09] R. Rouvoy, P. Barone, and Y. et al. Ding. MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. *LNCS*, 5525:164–182, 2009.

[RK09] T. Ruokolainen and L. Kutvonen. Managing interoperability knowledge in open service ecosystems. *Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOC*, pages 203–211, 2009.

[RLSS10] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. 44.1 cyber-physical systems: The next computing revolution. In *Proceedings - Design Automation Conference*, pages 731–736, 01 2010.

[RMjP15] M. A. Razzaque, M. Milojevic-jevric, and A. Palade. Middleware for Internet of Things : a Survey. *IEEE Internet of Things Journal*, 0(0):1–26, 2015.

[SB17] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. *Trends in Cognitive Sciences*, 2017.

[SBC+15] M. Serrano, P. Barnaghi, F. Carrez, P. Cousin, O. Vermesan, and P. Friess. Internet of Things - IoT Semantic Interoperability: research challeges, best practices, recommendations and next steps, 2015.

[SBKK17] S. Schmid, A. Bröring, D. Kramer, and S. et al. Käbisch. An Architecture for Interoperable IoT Ecosystems. *Lecture Notes in Computer Science*, 10218, 2017.

[SBV10] B. Sauser, J. Boardman, and D. Verma. Systomics: Toward a biology of system of systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 40(4):803–814, 2010.

[Sin18] R. R. Singh. *Designing for Multi-Agent Collaboration: A Shared Mental Model Perspective.* PhD thesis, University of Melbourne, 2018.

[Sma18] Interoperability in IOT based smart home: A review. *Review of Computer Engineering Studies*, 5:50–55, 2018.

[Ste17] A. Stein. Reaction Learning. In Christian Müller-Schloer and S. Tomforde, editors, *Organic Computing – Technical Systems for Survival in the Real World*, chapter Basic Methods, pages 287–328. Birkhäuser, Cham, 2017.

[SZZ+16] S. Soursos, I. P. Zarko, P. Zwickl, I. Gojmerac, G. Bianchi, and G. Carrozzo. Towards the cross-domain interoperability of IoT platforms. *EUCNC 2016 - European Conference on Networks and Communications*, pages 398–402, 2016.

[TC09] A. Tolk and Turnitsa C.D. An Extended Interoperability Framework for Joint Composability. In *Modeling, Simulation and Visualization Engineering Faculty Publications*, volume 52, 2009.

[TDT07] A. Tolk, S. Y. Diallo, and C. D. Turnitsa. Applying the Levels of Conceptual Interoperability Model in Support of Integratability , Interoperability , and Composability for System-of-Systems Engineering. *Journal of Systemics, Cybernetics and Informatics*, 5(5):65–74, 2007.

[TE06] E. Tamani and P. Evripidou. A pragmatic and pervasive methodology to web service discovery. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4278 LNCS:1285–1294, 2006.

[THIG11] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas. Service oriented middleware for the internet of things: A perspective. *Procedia Environmental Sciences*, 11, 01 2011.

[TKB10] A. Tiwana, B. Konsynski, and A. Bush. Research commentary platform evolution: Coevolution of platform architecture, governance, and environmental dynamics. *Information Systems Research*, 21:675–687, 12 2010.

[TMS17] S. Tomforde and C. Müller-Schloer. *Organic Computing - Technical Systems for Survival in the Real World.* Springer, 2017.

[Tom15] S. Tomforde. *Challenges and Solution Strategies for Mastering Interwoven Systems.* habilitation, Universität Augsburg, 2015.

[TPB+11] S. Tomforde, H. Prothmann, J. Branke, J. Hähner, M. Mnif, C. Müller-Schloer, U. Richter, and H. Schmeck. *Observation and Control of Organic Systems*, pages 325–338. Springer, 01 2011.

[TRBW16] S. Tomforde, S. Rudolph, K. Bellman, and R. Wurtz. An Organic Computing Perspective on Self-Improving System Interweaving at Runtime. *2016 IEEE International Conference on Autonomic Computing (ICAC)*, 245240:276–284, 2016.

[TV07] A. S. Tanenbaum and M. Van Steen. *Distributed Systems: Principles and Paradigms*, volume 1. Pearson Education, 2007.

[UM09] R. Urbanowicz and J. Moore. Learning classifier systems: A complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009, 09 2009.

[Vei03] D. Veit. *Matchmaking in Electronic Markets - An Agent-Based Approach towards Matchmaking in Electronic Negotiations*, volume 2882. Springer, 01 2003.

[VF13] Ovidiu Vermesan and Peter Friess. Internet of things: Converging technologies for smart environments and integrated ecosystems. In *River Publishers Series in Communications*. River Publishers (Aalborg Denmark 2013), 2013.

[VMZ10] C. Villalba, M. Mamei, and F. Zambonelli. A self-organizing architecture for pervasive ecosystems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6090 LNCS:275–300, 2010.

[WA13] D. Weyns and J. Andersson. On the challenges of self-adaptation in systems of systems. *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems - SESoS '13*, pages 47–51, 2013.

[Wei14] G. Weichhart. Requirements for supporting enterprise interoperability in dynamic environments. In *Proceedings of the I-ESA Conferences*, pages 479–488. Enterprise Interoperability VI, 01 2014.

[WF15] F. Wortmann and K. Flüchter. Internet of Things: Technology and Value Added. *Business and Information Systems Engineering*, 57(3):221–224, 2015.

[Wil09] J. Wilkes. Utility Functions, Prices, and Negotiation. *Market-Oriented Grid and Utility Computing*, pages 67–88, 2009.

[WvO12] M. Wiering and M. van Otterlo. *Reinforcement Learning - State of the art*, volume 1. Springer, 2012.

[XLZ04] P. Xuan, V. Lesser, and S. Zilberstein. Modeling Cooperative Multiagent Problem Solving as Decentralized Decision Processes. *Aamas*, 1:1–28, 2004.

[YAP12] E. Yahia, A. Aubry, and H. Panetto. Formal measures for semantic interoperability assessment in cooperative enterprise information systems. *Computers in Industry*, 63(5):443–457, 2012.

[ZTP14] M. Zdravkovi, M. Trajanovi, and H. Panetto. Enabling interoperability as a property of ubiquitous systems: towards the theory of interoperability-of-everything. *4th International Conference on Information Society and Technology (ICIST 2014) Kopaonik, Serbia*, (1):240–247, 2014.