FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Negotiating, Monitoring and Recommending Data Contracts in IoT Dataspaces

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Business Informatics

by

Florin Bogdan Balint, BSc
Registration Number 0725439

to the Faculty of Informatics

at the TU Wien

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

Vienna, 5th December, 2016

Florin Bogdan Balint	Hong-Linh Truong

Erklärung zur Verfassung der Arbeit

Florin Bogdan Balint, BSc
Spengergasse 27, 2902, 1050 Vienna

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

Wien, 5. Dezember 2016

Florin Bogdan Balint

Acknowledgements

First of all I would like to express my sincere appreciation and gratitude to my advisor, Mr. Prof. Hong-Linh Truong. He always gave me very valuable feedback, guidance, support and I really enjoyed working on this thesis.

Furthermore I would like to thank Prof. Christian Huemer, Prof. Sabine Theresia Köszegi and Prof. Markus Haslinger for their valuable feedback especially regarding the business and the legal aspects of this thesis.

I would also like to thank my family, my brother Adrian and my parents, my friends Marius, Bianca, Philipp, Rares and Sang Wha. Finally I would like to thank my girlfriend Simone for her support, patience and encouragement.

Abstract

With the technological advances from the past few years in the Internet of Things (IoT) the number of interconnected devices has increased drastically together with the amounts of produced data.

Multiple IoT platforms have been created to facilitate IoT data purchasing and selling. These platforms are limited regarding the data contract establishment and management. For example current platforms do not support the negotiation of individual data contracts, the monitoring of the data flow and making recommendations.

The challenge here is that different data contracts can be concluded for data produced by the same *Thing*. Another challenge is the monitoring of the data flow for data contracts: a data contract can consist of data produced by many *Things*, where each *Thing* can produce entirely different data.

This thesis introduces a new extensible IoT contract-aware framework, which supports the individual establishment of data contracts, the monitoring of the data flow for data contracts and making recommendations.

The methodological approach of this thesis is divided into three parts. In the first part literature research is conducted in order to find possible key elements regarding the design, monitoring and recommendation of data contracts. Based on these findings requirements are extracted for the implementation of a framework. In the second part a framework and a prototype as a proof of concept are implemented. Finally, in the last part, the implemented prototype is evaluated using two evaluation methods: a descriptive analysis to demonstrate the utility of the prototype and a dynamic analysis to demonstrate the performance and reliability of the prototype.

The obtained results demonstrate the effectiveness of the introduced framework for negotiating, monitoring and recommending data contracts in IoT dataspaces.

Kurzfassung

Mit den technologischen Fortschritten der letzten Jahre sind im Bereich des Internet der Dinge (IdD) die Anzahl an verbundenen Geräten, und die dadurch produzierte Datenmenge, drastisch gestiegen.

Mehrere IdD Plattformen wurden erstellt um den Kauf bzw. Verkauf von Daten zu ermöglichen. Diese sind jedoch eingeschränkt in Bezug auf der Vertragsgründung und Vertragsverwaltung. Zum Beispiel wird das Verhandeln von individuellen Datenverträgen nicht unterstützt, ebenso wie das Überwachen des Datenflusses und das Empfehlen neuer Verträge.

Die Herausforderung hier ist, dass unterschiedliche Datenverträge, für die produzierten Daten von einem *Ding*, abgeschlossen werden können. Eine weitere Herausforderung ist das Überwachen des Datenflusses für Datenverträge: Ein Vertrag kann aus Daten bestehen, die von beliebigen *Dingen* erstellt werden und jedes *Ding* kann unterschiedliche Daten produzieren.

Diese Arbeit stellt ein erweiterbares und skalierbares IdD vertragsbewusstes Framework vor, welches das Verhandeln von individuellen Datenverträgen, die Überwachung des Datenflusses für Datenverträge und Empfehlungen anbietet.

Der methodische Ansatz dieser Arbeit ist in drei Teilen eingeteilt. Im ersten Teil wird eine Literaturrecherche durchgeführt um mögliche Schlüsselelemente für das Erstellen, Überwachen und Empfehlen von Datenverträgen zu identifizieren. Basierend auf diesen Ergebnissen werden Anforderungen für die Implementierung eines Frameworks erstellt. Im zweiten Teil werden ein Framework und ein Prototyp als Konzeptnachweis erstellt. Im letzten Teil wird der Prototyp mittels zwei Methoden ausgewertet: einer deskriptiven Analyse, um den Nutzen des Prototyps zu demonstrieren und einer dynamischen Analyse um die Stabilität und Zuverlässigkeit des Prototyps zu beweisen.

Die resultierenden Ergebnisse beweisen die Effektivität des vorgestellten Frameworks bei dem Verhandeln, Überwachen und Empfehlen von Datenverträgen in IdD Datenräumen.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

API Application Programming Interface. xviii, 25, 41, 43

B2B Business-to-business. xviii, 10

BATNA Best Alternative to the Negotiated Agreement. xviii

DaaS Data as a Service. xviii, 15

DAO Data Access Object. xviii, 43

DSS Decision Support Systems. xviii, 11

ENS E-Negotiation System. xviii, 11, 12

ENT E-Negotiation Table. xviii, 11

GUI Graphical User Interface. xviii, 56, 67, 68

IdD Internet der Dinge. ix, xviii

IoT Internet of Things. vii, xiii, xviii, 1–5, 9, 10, 14, 15, 19, 21, 24–26, 29, 41, 43, 52, 55, 75

IT Information Technology. xviii, 14

JMS Java Message Service. xviii

JSF Java Server Faces. xviii

JSON JavaScript Object Notation. xviii, 34, 43, 67, 68

NAA Negotiation Agents-Assistant. xviii, 11

NoSQL non SQL. xviii, 55

NSA Negotiation Software Agent. xviii, 11

NSS Negotiation Support System. xviii, 11

QoD Quality of Data. xviii, 17, 33, 35, 37, 44, 48, 49, 71, 75

QoS Quality of Service. xviii, 17, 19, 33, 35, 37, 44, 48, 49, 71, 75

REST Representational State Transfer. xviii, 27, 56, 57, 67–69

SOA Service Oriented Architecture. xviii, 14

SOAP Simple Object Access Protocol. xviii, 14

SQL Structured Query Language. xviii

UML Unified Modeling Language. xviii, 12

URL Uniform Resource Locator. xviii, 57

WSDL Web Service Description Language. xviii, 13, 14

XML Extensible Markup Language. xviii, 14, 43

CHAPTER 1

Introduction

1.1 Introduction

The IoT is a "global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies" [4]. Companies and private research institutes have estimated that the IoT will consist of more than 20 billion interconnected devices by the year 2020 [5] [6] and the number of recorded data will double in size every two years [7].

With the increasing amount of data from the past few years, several IoT platforms and data markets (e.g. tilepay [8], MARSA [9], BDEX [10], Microsoft Azure Marketplace [11]) have arisen to facilitate data purchasing and selling. However, these platforms are limited regarding the data contract establishment and management.

In current developed platforms and IoT data markets, all customers mostly agree to the same contract when purchasing and selling data. For example, no current platform enables the negotiation of a data contract in such a way that one user can buy the data for one price and use it for computing and future predictions, but to forbid the publication of the results, and another user to buy the same data at the same time for another price, yet to have no such restrictions. Another limitation is that of monitoring the data stream for established data contracts. Finally, current platforms lack the functionality of recommending data contracts based on concluded contracts, by taking into account similar attributes (e.g. *Things* which were rated high by similar buyers).

1.2 Problem Statement

A dataspace can be seen as a collection of different data (different databases, different formats, etc.) [12]. We assume that within an IoT dataspace, data is produced by different *Things* and has certain properties (e.g., different quality, different market value price, etc.). This data can be packaged together to form a data package and sold with an individual data contract to a customer (see Figure 1.1). Additionally the same data can be uploaded directly into a cloud/database.

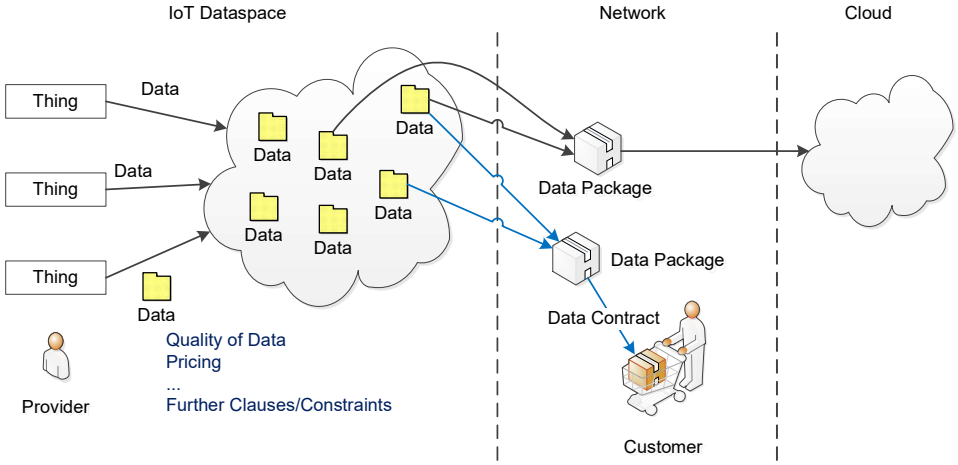


Figure 1.1: Concept of Data Contracts in IoT Dataspaces

Based on the previously described concept of data contracts in IoT dataspaces (see Figure 1.1), this thesis aims to answer the following research questions:

- What are possible key elements for the establishment of individual data contracts?
- What is a possible solution for monitoring the data flow of individually established data contracts? The main challenge here is that 1 to n different data contracts can exist and each data contract can consist of streamed data from 1 to m devices. Furthermore each device can produce entirely different data structures and broadcast data at different frequencies.
- What is a possible solution for making user recommendations based on previously concluded data contracts?

The previously mentioned research questions address aspects which can be in the interest of multiple stakeholders. Being able to establish individual data contracts with different contractual clauses (e.g., different price, different usage rights) might increase the number of sales, which in turn increases the profit. Furthermore, monitoring the data flow of a data contract enables the detection of anomalies (e.g., device broadcasting malfunction).

This way a *Thing* provider can reconfigure hist/her devices, so that it produces the expected data accordingly. Finally, when dealing with high amounts of devices (e.g., smart city with thousands of *Things*), it might be helpful to have a recommender system, which recommends the purchase of data produced by *Things* that align with a users' interests.

The goal of this thesis is to develop a new framework from a business perspective, which supports the establishment of individual data contracts, the monitoring of streamed data according to concluded data contracts and recommendations for *Things* within one IoT dataspace.

For the scope of this thesis the following assumptions have been made:

1. *Things* which produce data already exist and the focus is on the data contract management part.
2. Interested parties are willing to negotiate data contracts for data produced by *Things*.
3. When relating to data produced by *Things*, we assume that all data is non-personal. Dealing with personal data would go beyond the scope of this thesis.

1.3 Methodological Approach

The methodological approach of this thesis is divided into three main parts: analysis, design and development and evaluation. In the first part, literature research is conducted to find possible key elements regarding the design, monitoring and recommendation of data contracts. The literature research includes legal and business aspects which are related to concluding IoT data contracts. Based on these findings, requirements are extracted for the design of a new framework and the implementation of a prototype.

In the second part, the design and development part, a framework is designed and a prototype is implemented, based on the previously defined requirements. This prototype serves as a possible solution for the previously mentioned problem.

Finally, in the last part, the prototype is evaluated using two evaluation methods. The first evaluation method analyses the prototype from an end user perspective and is a descriptive analysis based on a few representative scenarios to demonstrate the utility of the prototype. The second evaluation method analyses the prototype from a technical perspective and is a dynamic analysis with the focus on performance.

1.4 Contribution

This thesis introduces a new extensible IoT contract-aware framework, which supports the individual conclusion of data contracts, the monitoring of the data flow for established contracts and individual recommendations based on previously concluded data contracts.

Some parts of the work and results of this thesis have been extracted and summarized into the following article: "On Supporting Contract-aware IoT Dataspace Services" [13]. This article was be submitted to the 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering April 6 – April 8, 2017 – San Francisco, USA.

1.5 Thesis Structure

The rest of this thesis is structured as follows:

- Chapter 2 presents relevant theoretical aspects regarding IoT data contracts, their negotiation, recommendation techniques, relevant legal aspects and current state of the art projects.
- In Chapter 3 requirements are presented, which have been extracted based on the previously conducted literature research.
- In Chapter 4 the new proposed extensible contract-aware framework is described.
- In Chapter 5 the implemented prototype, which servers as a possible solution for the previously mentioned problem, is presented and evaluated.
- Finally the findings of this thesis are summarized and future work is outlined in Chapter 6.

CHAPTER 2

Background and State of the Art

In this chapter we are going to identify possible key elements for the establishment of individual data contracts. First we are going to provide an overview of important legal aspects regarding the conclusion of data contracts. Second we are going to analyse relevant theoretical aspects regarding data contracts, their negotiation, monitoring and recommendation techniques. Finally we are going to conclude this chapter with a short overview of state of the art IoT data markets, their capabilities and their limitations.

2.1 Legal Perspective of Data Contracts

When establishing IoT data contracts, depending on the jurisdictional area, different laws may apply. In this section we are going to elaborate upon legal aspects and possible legal constraints, which have to be taken into account when concluding such contracts. First we are going to analyse a few legal aspects from the perspective of European law and afterwards from the perspective of Austrian law. The legal aspects are only analysed from these two perspectives, because an analysis from an international perspective would go beyond the scope of this thesis.

2.1.1 European Law

In the European Union each member state has its own national law. The European Union tries to harmonize the national laws of the different countries through various treaties, regulations and directives. With Article 6 of the consolidated version of the treaty of the European Union, the Union recognises the rights, freedoms and principles set out in the Charter of Fundamental Rights of the European Union. These rights have the same legal value as the Treaties [14].

In Article 7 of the Charter of Fundamental Rights of the European Union it is stated that “everyone has the right to respect for his or her private and family life, home

and communications" and in Article 8 it is stated that "everyone has the right to the protection of personal data concerning him or her" [15].

Multiple directives have been proposed in order to protect personal freedom, data and privacy, which either have already been transposed into national law by the the EU Member States or have to be transposed into national law in the next few years. In this part of the chapter some of those regulations are presented.

Data Protection Directive 2016/680

It has been assumed that only non-personal data is bought/sold, however if a company that sells non-personal data is possessing personal customer data (e.g., customer name, address, phone number, payment details, etc.), then this data is subject to the data protection law.

The Data Protection Directive 2016/680 [16] imposes regulations on the protection of personal data. The directive entered into force on 5 May 2016 and the EU Member States have to transpose it into their national law by 6 May 2018 [16, 17].

This directive imposes principles of processing personal data. Personal data (i.e., any information relating to an identified or identifiable natural person 'data subject') has to be processed lawfully, accurate, kept in a form which permits identification and processed in a manner that ensures appropriate security and there has to be a controller (i.e., a competent authority) that shall be responsible for the compliance [16, Art. 4].

Furthermore, member states have to provide time limits for the erasure of personal data [16, Art. 5]. The processing of special categories of personal data (e.g., ethnic origin, political opinions, religious or philosophical beliefs, etc.) is only allowed where strictly necessary and subject to appropriate safeguards for the rights and freedoms of the data subject [16, Art. 10].

Member states have to provide certain information that has to be made available to the data subjects. Amongst others these include the contact details of the controller, contact details of the data protection officer, the legal basis for the processing and the period of the data storage [16, Art. 13].

Data subjects have the right to obtain information whether or not personal data is being processed and the purpose of and the legal basis for the processing. Furthermore data subjects have the right to request from the controller rectification or erasure of personal data and right to lodge a complaint with the supervisory authority (i.e., independent public authority which is established by a Member State) and the contact details of the supervisory authority [16, Art. 14].

Regarding the security of the data processing, measures have to be implemented to prevent unauthorized access, reading, copying, modification or removal of data. Furthermore member states have to ensure that persons authorized to use an automated processing system only have access to the data covered by their access authorization [16, Art. 29].

In case of a personal data breach the supervisory authority has to be notified without undue delay and (where feasible) within 72 hours. This notification has to contain the nature of the data breach, the categories and an approximate number of data subjects and data records which are concerned [16, Art. 30]. If this data breach is "likely to result in a high risk to the rights and freedoms of natural persons" it has to be communicated to the data subjects without undue delay [16, Art. 31].

E-Commerce Directive 2000/31/EC

The E-Commerce Directive 2000/31/EC entered into effect on 17th July 2000 and has the objective to ensure "the free movement of information society services" [18, Art. 1].

Service providers (i.e., any natural or legal person which is providing an information society service) have to provide general information which has to be easily accessible to the recipients of the service. Amongst others this information has to include the name, the address and the e-mail address of the provider [18, Art. 5].

This directive enforces the possibility to conclude contracts by electronic means (with a few exceptions like creating or transferring rights in real estates) [18, Art. 9]. Furthermore the service providers have to provide information to the recipient of the service regarding the different technical steps that have to be followed in order to conclude a contract, contract terms, general conditions and any relevant codes of conduct [18, Art. 10].

Once an order is placed, the service provider has to acknowledge the receipt of the recipient's order without undue delay and by electronic means. Additionally, technical means have to be made available to identify and correct input errors, prior to the placing of the order [18, Art. 11].

If an information society service is provided, that consists of the transmission in a communication network of information or of storage of information, provided by a recipient of the service, there is no general obligation to monitor the information these services transmit or store [18, Art. 15].

Directive on the supply of digital content

There has been a proposal for a directive of the European Parliament and of the European Council on certain aspects concerning contracts for the supply of digital content (i.e. data which is produced and supplied in digital form, for example video, audio, applications) [19]. At the time of the conducted research, this directive has not been accepted or adopted yet.

After concluding a contract the supplier has to supply the digital content immediately to the consumer or to a third party which allows the consumer to access the digital content [19, Art. 5]. In case the content is not supplied immediately, the consumer is entitled to terminate the contract immediately [19, Art. 11].

The digital content has to conform with the concluded contract and where relevant to be of the quantity, quality, duration, version, and other features such as accessibility,

continuity as required by the contract. In case this is not specified, the digital content has to be fit for the purposes for which digital content of the same description would normally be used [19, Art. 6].

Regarding the conformity of the contract, the burden of proof is on the supplier. However, this does not apply if the environment of the consumer is not compatible with interoperability and other technical requirements if the supplier informed the consumer of such requirements beforehand. Furthermore to establish the consumer's digital environment, the supplier and the consumer have to cooperate. In case the consumer fails to cooperate the burdon of proof will be on the consumer [19, Art. 9].

The supplier is liable to the consumer for any failure or lack of conformity to supply the digital content, even if it is supplied over a period of time [19, Art. 10]. In case the digital content is not conform to the contract, the consumer is entitled to have the digital content brought into conformity with the contract free of charge (excepting that it is impossible, disproportionate or unlawful) within a reasonable time. If the digital content is provided in exchange for a payment, the consumer is entitled to a proportionate reduction of the price (in accordance to the decrease of the value of the digital content), if the supplier has not completed the remedy, or if it is impossible. In some cases the consumer is entitled to terminate the contract (e.g., if the lack of conformity with the contract impairs functionality) [19, Art. 12].

Another aspect covered by this directive is that of contract termination. In case of a contract termination the supplier has to reimburse the paid price without undue delay and no longer than 14 days from the receipt of notice. Additionally the supplier can prevent any further use of the digital content by the consumer, however the consumer does not have to pay for any use made in the period prior to the termination of the contract [19, Art. 13].

The supplier is liable to the consumer in case of any economic damages to the digital environment of the consumer, which are caused by the lack of conformity with the contract or a failure to supply the content [19, Art. 14].

This directive regulates the modification of the digital content of a contract during a subscription period. If the digital content is supplied over a period of time, the provider may alter the functionality, accessibility and other features (e.g., continuity) of the digital content only if the contract stipulates so, or the consumer is notified in advance and the consumer can terminate the contract free of charges within no less than 30 days from the receipt of the notice [19, Art. 15].

If a contract is concluded for an indeterminate period or the period exceeds 12 months, the consumer is entitled to terminate the contract after the expiration of the first 12 months. The termination in this case has to become effective 14 days after the receipt of the notice. In the case that the digital content was supplied for a price, the consumer remains liable to pay the price for the supplied content for the period of time before the termination of the contract [19, Art. 16].

Jurisdiction

IoT data contracts can be concluded between two parties from two different countries. In this case the parties may choose the law that applies for the contract. The choice has to be made expressly or clearly [20, Art. 3].

If the law that applies for a contract has not been explicitly agreed upon or mentioned, in case of the sell of goods a contract "shall be governed by the law of the country where the seller has his habitual residence" and in the case of the sell of services "by the law of the country where the service provider has his habitual residence" [20, Art. 4].

Furthermore in case of a legal dispute according to the Art. 4 of Regulation (EU) No 1215/2012 "persons domiciled in a Member State shall, whatever their nationality, be sued in the courts of that Member State" [21].

2.1.2 Austrian Law

In Austria the previous European Directives were implemented into national law. The Data Protection Directive 95/46/EG was implemented into national law with the Data Protection Act 2000 (i.e., Datenschutzgesetz 2000) [22] (the new Directive (EU) 2016/680 not being implemented yet at the time of the conducted research). The E-Commerce Directive was also implemented into national law with the E-Commerce Act [23].

IoT devices produce data and this data might be processed and enriched (e.g., pictures of the environment or a temperature sensor records the temperature and measurement errors might be automatically corrected, where inexistent predicted and further processed). In this case intellectual property rights might apply. According to Austrian Law, databases are viewed as a collection of works or collective works, data or other independent elements, which are systematically or methodically arranged. A collective work is protected by copyright laws if it is an intellectual creation [24, Art. 40f].

Furthermore the Austrian Civil Code (i.e., ABGB) applies. Different prices may be charged for the same good, however "laesio enormis" has to be considered when doing so. "Laesio enormis" states that when a contract is concluded for a product and the price is more than twice as much as what is usually paid for such a product, the contract can be terminated by the customer [25, Art. 934]. For example if a data contract is concluded for weather data and this weather data usually costs 10 EUR, but the data is sold for 20 EUR or more, "laesio enormis" applies and the contract can be terminated immediately.

In this section we have shown that multiple directives exist on a European level that try to harmonize the law across the member states. Some of those regulations, which have already been transposed into national laws, and further national laws, need to be taken into account when officially concluding IoT data contracts (e.g., acknowledge the receipt of an order), while others exist to avoid any ambiguities (e.g., jurisdiction, contract termination, etc.).

2.2 E-negotation of Data Contracts

In the previous section we analysed certain legal aspects that have to be taken into into consideration when concluding IoT data contracts. Since we assumed that a data provider and an interested party are willing to negotiate a data contract (see Chapter 1), in this section we are going to focus on the negotiation part of data contracts. At the beginning, we are going to provide a short introduction into negotiations and e-negotiation systems. Afterwards we are going to analyse negotiation system models among with key functions for the support of negotiations. Finally we are going to take a look at current architectural styles of e-negotiation systems.

2.2.1 Background

Business-to-business (B2B) e-commerce processes follow generally a three phase model. In the first phase, an interested party will look for business partners and try to come to an acceptable agreement. The partners might bargain about the price and other contractual terms and therefore the second phase is that of negotiating an agreement. If the negotiation is successful a contract is established, which the parties have to process in a third phase (e.g., logistics, payment, etc.) [26].

During a negotiation two or more parties make decisions and interact with each other in order to achieve mutual gain [27]. During a negotiation process, a proposal is created and sent to another party and a new proposal might be generated after receiving a counter proposal. This process continues until either an agreement or a deadlock is reached [28]. Traditionally negotiations end with binding contracts as an outcome [29].

Gregory Kersten [1] identified amongst others the following components that the practice and theory of negotiations include: the negotiation problem (i.e., what is being negotiated), objectives (i.e., what one wants to achieve), communication (e.g., formal, informal), the negotiation process (e.g., arguments, offers, messages) and the negotiation outcomes (e.g., compromise, implementation).

Negotiations can be classified in various ways. One way used by Gregory Kersten [1] is by taking into account the number of parties (see Figure 2.1). From this perspective four types of negotiations can be distinguished. In the first type, the bilateral type, the participation is restricted to two parties and the negotiation process is often well defined for a single issue. The second type of negotiation is the multi-bilateral type. In this case the negotiation involves two sides: one side is represented by a single party and the other side is represented by many parties. The third type is multilateral negotiations which involves many parties and the negotiation process is rarely well defined. Finally the last type is that of nested negotiations which involves two or more parties and the negotiations are conducted on behalf of organizations or individuals.

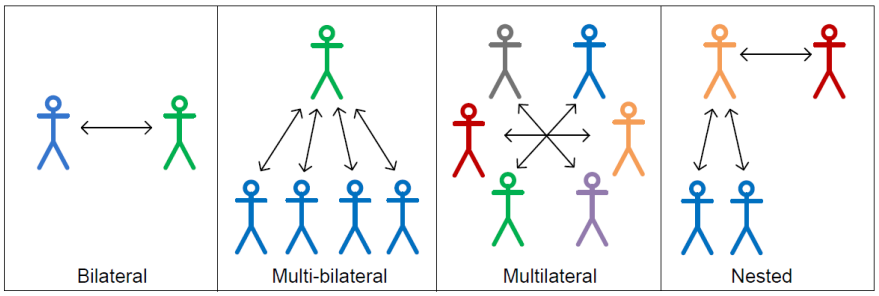


Figure 2.1: Negotiation types overview [1]

2.2.2 E-Negotiation and E-Negotiation Systems

An E-Negotiation System (ENS) is a "software that employs internet technologies and is deployed on the web for the purpose of facilitating, organizing, supporting and/or automating activities undertaken by the negotiators and/or a third party" [2].

Various kinds of software systems have been designed to support individuals in negotiations (see Figure 2.2). An E-Negotiation Table (ENT) is a software which provides negotiators with a virtual bargaining table. In its simplest form it supports the posting of offers and sending of messages. A Negotiation Software Agent (NSA) is a software which is involved in the negotiations, can make decisions on behalf of one party and its purpose is to automate one or more negotiation activities. A Negotiation Agents-Assistant (NAA) is a software that provides timely and context-specific advice, critique and support. Decision Support Systems (DSS) are systems that support the decision making process. As can be seen in Figure 2.2, DSS tie the previously mentioned systems together. Negotiation systems provide support which specifically deals with the negotiation process and extend the functionalities provided by a DSS [2].

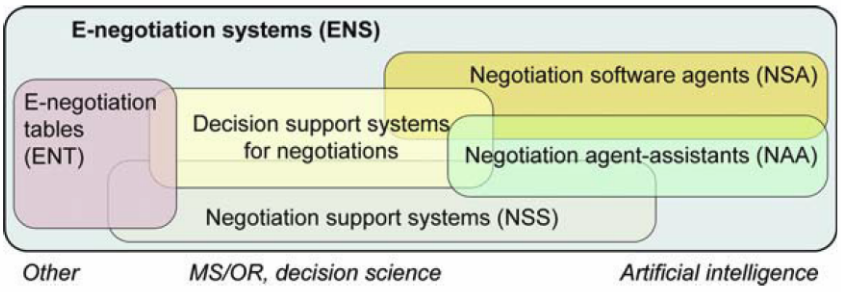


Figure 2.2: E-negotiation systems overview [2]

The key aspect of a Negotiation Support System (NSS) is that the decision process it supports is consensual [2]. Kersten & Lai [2] identified a few properties regarding the users, that need to be taken into account when designing an NSS. They stated that users

are: independent, representing their own interests, able to reject every offer, request another offer and propose a counteroffer.

Regardless of the negotiation type, each negotiation is based on a protocol which specifies the users interactions. Even if it is not explicitly specified, every negotiation system is using a negotiation protocol, which can be derived from possible interactions between the negotiators and the system [30]. A negotiation protocol can also be seen as a model, which guides the user and imposes restrictions on their activities. An e-negotiation protocol is thus required to specify and control the activities undertaken by the negotiators [31].

2.2.3 Models and Functions

In order to be able to develop an ENS which supports the negotiation of various data contracts, it is important to know how data contracts can be described generally and what key functions such a system should support.

Schoop et al. [32] identified three main models regarding electronic negotiations: electronic auctions, negotiation agents and negotiation support. The first two models are quantitative approaches because they aim at enabling an efficient negotiation process in terms of finding an economic optimum. Electronic auctions work according to general auction principles (i.e., bids concerning different criteria, such as price, are placed on a good to be purchased or sold). Negotiation agents take over parts of or the whole negotiation process for the human principal and negotiation support approaches provide support for complex negotiations, leaving the control over the negotiation process with the human.

D.K.W. Chiu et al. [28] proposed a meta-modeling approach based on a meta-model which is universally applicable. In this meta-modeling approach e-contracts can be specified based on a standard contract template. A contract template is a reference document, based on which new contracts can be negotiated. A contract template consists of a number of contract clauses, each addressing a specific concern. Each contract clause contains a set of template variables which can be negotiated [28]. D.K.W. Chiu et al. propose the negotiation of data contracts based on the negotiation of template variables and give the following example for a contract template:

"The PURCHASER shall send a Letter of Credit for the GOODS to the SUPPLIER in the currency of [ ] within [ ] days of the invoice date..." [28].

D.K.W. Chiu et al. furthermore presented a meta-model of an e-contract template in Unified Modeling Language (UML). A contract involves at least two parties. A contract template consists of a number of contract clauses which are subject to the negotiation, and these contract clauses typically include obligation, permission and prohibition [33].

For the general development of an ENS, Braun et al. [3] proposed a five stage model (see Figure 2.3):

1. The planning phase in which negotiators formulate their representation of the negotiation problem including issues specifications and options.

2. Agenda setting and exploring the field which includes the negotiated issues and their meaning. The result of these discussions may result in adding or removing options and that the negotiators may have to revise the problem, objectives and preferences.

3. Exchanging offers and arguments in which the parties learn about each others limitations and identify the key issues and critical areas of disagreement.

4. Reaching an agreement is when the parties realize that the negotiation has been successful, having identified all critical issues.

5. Concluding the negotiation which takes place when the parties reach an agreement.

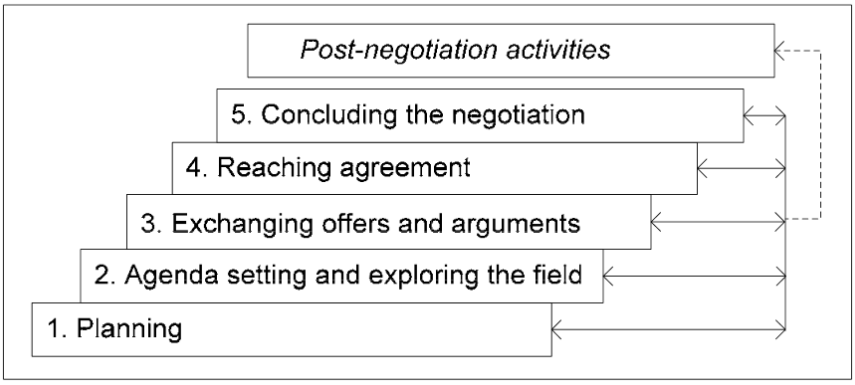


Figure 2.3: Five Stage Model [3]

2.2.4 Architecture

Several architectures with the focus from different perspectives have been proposed for the developing negotiation support systems. One common point for some of these proposed architectures regarding the addressability is that of the usage of web services based on a universal Web Service Description Language (WSDL) [34] [35].

Benyoucef & Keller [36] proposed a conceptual architecture for combined negotiation support systems. A combined negotiation is a negotiation when a consumer is interested in many goods or services and engages in many negotiations at the same time. The negotiations can be of any type (e.g., fixed price sale, bilateral bargaining, etc.) and are typically independent. The proposed architecture is divided in two parts: a start-up time architecture during which the workflow is constructed and a run-time architecture during which the combined negotiation is conducted. In comparison to the start-up

time architecture, the run-time architecture contains multiple agents and each agent communicates individually with a negotiation server.

Benyoucef & Rinderle [37] proposed a framework based on a service oriented architecture for e-negotiation systems consisting of two main components: an e-marketplace and an automated negotiation system. The e-marketplace enables a designer to specify negotiation protocols. This component communicates through web services with the automated negotiation system. The framework is based on a Service Oriented Architecture (SOA) to enable communication between different Information Technology (IT) components and to provide a standardized way of communication. The web services are described via WSDL and accessed using Simple Object Access Protocol (SOAP) requests.

Schoop et al. [32] developed *N*egoisst, a process-oriented negotiation support system aiming at enabling intuitive, unambiguous, and efficient electronic negotiations between human negotiators. *N*egoisst was written in Java and implemented in a client–server architecture. At the server side a three-tier-client architecture was implemented consisting of a presentation layer, an application layer and a data layer.

The presentation layer provides a web user interface which is accessible through a browser. Here a user can register, start a new negotiation, send messages and reply to incoming messages, etc. The application layer consists of two main components: a negotiation component which is responsible for starting, storing, and proceeding a negotiation and a satisfiability checks component, which monitors the negotiations and checks the contract fulfilment. The data layer is responsible for storing all the relevant information. In this case two types of data storages were used: (1) Extensible Markup Language (XML) documents which are representing the contracts and contain all relevant information and (2) a relational database for the rest.

In this section we have analysed theoretical and technical aspects of negotiations and e-negotiations which can be applied for the establishment of individual data contracts in IoT dataspaces. Based on the scenario, different negotiation types (see Figure 2.1) might be of interest for the negotiation of data contracts. For example in the case of one *Thing* provider with a few devices, who would like to sell the data to a few customers, bilateral negotiations can be used for the establishment of individual data contracts. In the case of two or more companies interested in establishing data contracts for the purchase/selling of different data nested negotiations could be applied. To support and ease the negotiation process, data contract templates could be used. Finally, the negotiation process can be implemented by taking into account the Five Stage Model (see Figure 2.3) (e.g., starting from the selection of the data, agenda setting, exchanging arguments, reaching an agreement).

2.3 IoT Data Contracts

In our everyday life we conclude contracts all the time. For example, when we buy something from a vending machine we conclude a contract with the vending machine provider. In this simple case all contractual terms are clear to all parties: the price, which is usually clearly visible, and the product (e.g., a bar of chocolate), which is also clearly visible. When we conclude a more complex contract (e.g., a software hosting services agreement), multiple aspects have to be taken into account and mentioned explicitly in order to avoid ambiguity (e.g., price, fees, handling expenses, penalties, maintenance, support etc.). In this section of this thesis, we are going to analyse IoT data contracts with the goal to identify possible data contractual clauses, which can also be negotiated, so that individual data contracts can be established.

2.3.1 Contracts and Data Contracts

"A contract is a binding agreement between two or more parties defining a set of obligations and rewards in a business process" [28] and can also be seen as an "agreement between parties to create business relations and meet legal obligations" [38]. Data contracts have already been proposed by Truong et al. [39] to describe data contractual terms when data is purchased or sold.

2.3.2 Data Rights

One possible property, that can be taken into account when establishing a data contract, is that of data rights. Truong et al. [39] [40] investigated and identified the following data rights as properties of data contracts, by studying existing data license agreements and service contracts, which are relevant from the perspective of contracts for Data as a Service (DaaS): derivation, collection, reproduction, attribution, commercial and non-commercial usage.

One problem, that was identified by Kumari et al. [41] is that once the data has been delivered, it cannot be controlled any more by the data provider. One possible way to deal with this problem is from a legal perspective, by taking into account certain legal aspects, which leads to our next possible data contractual properties: regulatory compliance and control and relationship.

2.3.3 Regulatory Compliance and Control and Relationship

When concluding service contracts several legal aspects must be taken into consideration. Regulatory compliance refers to the protection of privacy and confidentiality of information [39]. Multiple compliance laws and directives already exist to protect the privacy and the confidentiality of data (e.g., Sarbanes–Oxley Act of 2002 [42] and the European Union Data Protection Directive 2016/680 [16]).

Regarding the control and relationship, Bradshaw et al. [43] conducted a comparison and analysis of terms and conditions for cloud computing services. Some of the terms and

conditions align with the research conducted by Truong et al. [39]: warranty, indemnity, liability and jurisdiction.

2.3.4 Pricing

The pricing is another data contractual property which can be taken into consideration for the establishment of data contracts. In some cases the price is not relevant (e.g., accessing public data for free), but for the rest of this thesis we will assume that a data provider would always like to sell the data produced by his/her *Things* for a certain amount of money.

Various pricing models and pricing model categories exist. Sen et. al., [44] conducted a survey regarding smart data pricing and identified two main pricing model categories: static and dynamic pricing. Static pricing models are models in which the price does not vary over time. Amongst others, Sen et. al., [44] identified the following most common static models: usage-based (i.e., depending on the data volume), flat-rate (i.e., unlimited access for a certain period of time), priority pricing (i.e., better quality of service for certain packages) and time-of-day pricing (i.e., different prices for different usage hours).

Another possibility to set the price is by using a dynamic pricing model. Dynamic pricing models are models in which the price can vary over time and allow more flexibility [44]. Examples of dynamics pricing models are auctions (i.e., bid each time), congestion pricing (based on real-time or current congestion) and day ahead pricing (guarantee a certain price in advance) [44].

2.3.5 Purchasing Policy

Additional data contractual clauses which might be included when concluding data contracts are with respect to the purchasing policy. Goodchild et al. [45] presented a new specification of Business-to-Business contracts in which such clauses have been explicitly specified. Furthermore, some of these clauses can also be found in the general terms and conditions of the platforms mentioned in Chapter 2.6:

- Contract Termination: Term specifying contract termination details.
- Shipping / Delivery: Term specifying how the products/goods (i.e., data) will be shipped or delivered.
- Refund: Term specifying if price charges are refundable or not.

2.4 Monitoring Data Contracts

In the previous section we identified and elaborated upon a few possible data contractual clauses (e.g., data rights, pricing, etc.). In this section we are going to analyse two further data contractual properties: Quality of Data (QoD) and Quality of Service (QoS). These properties can be used for monitoring the data flow of individually established data contracts.

2.4.1 Quality of Data

Data represents real world objects, in a format that can be stored, retrieved an elaborated by a software procedure. QoD has become a significant issue for operational processes of businesses and organizations [46]. The Data Warehousing Institute estimated that data quality problems cost U.S. businesses more than USD 600 billion a year [47]. For this reason we consider data quality as one possible data contractual term. This has already been proposed by Truong et al. [39], [40].

The challenge here is to measure the data quality: within an IoT dataspace multiple *Things* can produce different types of data and each produced data can have different quality aspects. Multiple dimensions have been associated with respect to data quality [46, 47, 48, 49, 50, 51]. In this section we are going to elaborate upon some of these dimensions.

Completeness

Data completeness is generally defined as "the extent to which data are of sufficient breadth, depth, and scope for the task at hand" [48] and can be further divided into different types [49]:

- Schema completeness: represents "the degree to which entities and attributes are not missing from the schema" [46, p. 24].
- Column completeness: a function of "the missing values for a specific property or column in a table" [46, p. 24].
- Population completeness: measurement for "the missing values with respect to a reference population" [46, p. 24].

Carlo Batini and Monica Scannapieco [46] estimated that both data and schema dimensions are important, because low quality data influences the quality of business processes and a low quality schema (e.g., an unnormalized schema) in the relation model results in potential redundancies and anomalies during the lifecycle of data usage. In Table 2.1 an example of completeness error is illustrated at the fourth entry: the scale of the recorded temperature is missing.

ID	Temperature	Scale	Date
1	2.5	Celsius	2016-12-01 12:00:00
2	3.8	Fahrenheit	2016-12-01 14:00:00
3	3.6	Celius	2016-12-01 14:30:00
4	2.0	*null*	2016-12-01 20:00:00

Table 2.1: QoD Problems Example - Temperature Recordings

Consistency and Conformity

Consistency is the "degree to which data managed in a system satisfy specified constraints or business rules" [51]. Such rules can be integrity constraints (e.g., uniqueness of customer identifiers) or referential integrity (e.g., for each order, a customer record must exist).

"Data conformity describes how well data adheres to standards and how well it's represented in an expected format" [52]. C. Tărâță [53] stated that conformity consists in ensuring that data is following a standard format (e.g., date values as *yyyy/MM/dd*, or customer genders can only be *male* or *female*).

Accuracy

"*Accuracy* is defined as the closeness between a value v and a value v' considered the correct representation of the real-life phenomenon that v aims to represent" [46, p. 20].

Batini and Scannapieco [46] and Fürber and Hepp [50] distinguish between syntactic accuracy and semantic accuracy. Syntactic accuracy refers to syntax violations and it refers to the domain definition. If the true value is v=*Jack* and the real value is v'=*John* the data entry is syntactically correct because both values are from the same domain. The semantic accuracy refers to the closeness of the value v to the true value v'.

In Table 2.1 an example of a semantic accuracy error is provided in the second entry, where "Fahrenheit" is listed as the scale of the temperature recording which is not accurate, the correct value would be "Celsius". The third entry represents a syntactic accuracy error, because "Celius" is not a correct temperature scale.

Time Related Dimensions

One aspect of data is that it changes and it is updated over time [46]. Currency concerns how promptly data is updated and can be "measured with respect to the *last update* metadata". Simple currency measurement values are *current* and *not current*. For example if an entry is estimated to change every five years and the last update of was one year ago, then the respective data can be assumed to be *current*. However, if the last update was performed 10 years ago the data can be assumed to be *not current*. The currency can also be computed as follows [46]:

$$Currency = Age + (DeliveryTime - InputTime) \tag{2.1}$$

"*Volatility* characterizes the frequency with which data vary in time" [46, p. 29]. The birth date of a person has a volatility equal to 0, as it does not change at all. An example of high volatility would be that of stock quotes, which changes frequently. Batini and Scannapieco [46] proposed to use the length of time data remains valid as a metric for volatility.

"*Timeliness* expresses how current data are for the task at hand" [46, p. 29]. For example the timetable for a university course can be current by containing the most recent data, however if this data is provided after the start of the classes it is not timely [46]. The timeliness is defined as:

$$Timeliness = \max\{0, 1 - \frac{currency}{volatility}\} \tag{2.2}$$

Other Data Quality Dimension

Batini and Scannapieco [46] proposed two further dimensions regarding the geographical and geospatial domain: positional accuracy and attribute / thematic accuracy. They defined positional accuracy as the quality parameter indicating the accuracy of geographical positions and the attribute / thematic accuracy as the positional and/or value accuracy of properties such as sociodemographic attributes in thematic maps. The Victorian Spatial Council [54] based on the ISO Standard 19113 further divided the positional accuracy into absolute or external accuracy (i.e., closeness of reported coordinate values to values accepted as or being true) and relative or internal accuracy (i.e., closeness of the relative positions of features in a dataset to their respective relative positions accepted as or being true).

2.4.2 Quality of Service

Another aspect that can be taken into account when concluding data contracts is that of the QoS. For example sensors are expected to measure different things and to broadcast the measured values with a certain frequency. Furthermore, it might be in the interest of the involved parties of a data contract to find out, if the expected broadcasting frequency does not equal the actual broadcasting frequency. For this reason we consider the QoS as one possible data contractual property.

QoS monitoring has also become a key activity, especially since assessing the actual quality of what service users are paying for has become a mission-critical business practice requirement [55]. R. Duan, X. Chen and T. Xing stated that the QoS of application and service layers is perceived directly by customers and therefore becomes the most important standard of evaluation for IoT [56].

Multiple properties have already been associated with the QoS: D. A. Menasce stated that the quality of service is a combination of several qualities or properties of a service, such as [57]: availability, security (which include authentication mechanisms), response time and throughput (i.e., the rate at which a service can process requests). Other characteristics have been identified from a network perspective [58, 59]: latency or delay

(i.e., the time that has passed while the data was transmitted from source to destination) and jitter (i.e., is the variation in delay).

In this and the previous section of this thesis we discussed different aspects, which have already been proposed as possible data contractual clauses through the literature, however not as a combination. Therefore we propose the following set of representative data contractual clauses as one possibility to model data contracts:

- **Data Rights**: Derivation, Collection, Reproduction, Commercial Usage
- **Quality of Data**: Completeness, Conformity, Average Message Age, Average Message Currency
- **Quality of Service**: Availability
- **Pricing Model**: Price, Subscription Period
- **Purchasing Policy**: Contract Termination, Shipping, Refund
- **Control and Relationship**: Warranty, Indemnity, Liability, Jurisdiction

2.5 Recommending Data Contracts

In the previous sections we elaborated upon data contracts and their establishment. Within an IoT dataspace a lot of *Things* which produce data can exist. Potential buyers might face the difficulty of having to choose between different devices from different dataspaces. In this situation it might be helpful to have a recommender system, which recommends the purchase of data produced by *Things* that align with a users interests. In this section we are going to provide an overview of recommender systems together with a few basic methods used for building recommender systems.

Recommender Systems are software tools and techniques that provide suggestions for items to be of use to a user and to support the decision-making process. An item in this context is a general term referring to what the system recommends to users [60].

Ricci et al., [60] stated that there are two types of recommendations: personalized recommendations and non-personalized recommendations. Personalized recommendations are recommendations for different users or user groups. In this case users receive diverse suggestions and recommender systems try to predict what the most suitable products or services are, based on the user's preferences and constraints. In order to do so recommender systems collect user preferences, which are either explicitly expressed, e.g., as ratings for products, or are inferred by interpreting user actions. Non-personalized recommendations are recommendations which are easier to generate (e.g., top ten selection of books) and are not user-specific.

2.5.1 Collaborative Filtering Recommender Systems

Schafer et al., [61] defined collaborative filtering as the process of filtering or evaluating items using the opinions of other people. A very simple example is when people discuss books they have read and restaurants they have tried. The task of a collaborative filtering algorithm is to predict the utility of items to a particular user based on user votes from a sample or population [62].

Ratings in Collaborative Filtering

Two types of ratings were distinguished by Schafer et al., [61]: explicit ratings and implicit ratings. Explicit ratings require a user to rate a product and offer an accurate description of a user's preference. The main disadvantage of explicit ratings is, that users have to rate items. Implicit ratings are observations of user behaviour, from which preferences are derived, e.g., "time spend reading about a product". These ratings however might not be accurate, e.g., "time spend reading" can be drastically extended if the respective person takes a lunch break and the product information is still displayed on the screen [61].

According to Schafer et al., [61] the most common types of ratings are the following: unary (e.g, *good* or *don't know*), binary (e.g., *good* or *bad*) and integer *Likert*-like (e.g., $1 - 5$). Furthermore Schafer et al., [61] separated collaborative filtering algorithms into

two types: non-probabilistic and probabilistic algorithms. An algorithm is probabilistic, if it is based on an underlying probabilistic method.

Non-probabilistic Algorithms

Neighbourhood collaborative filtering use user-item ratings stored in the system directly to predict ratings for new items. This can be achieved using either user-based or item-based recommendation [63].

User-based Nearest Neighbour Algorithm

A very known class of nearest neighbour algorithms is that of user-based algorithms [61]. This approach relies on the rating of user u for items i, and on the ratings of other users as well. Desrosiers and Karypis [63] stated that the rating of a user u for a new item i is likely to be similar to that of another user n, if u and n have rated other items in a similar way. If a user n is similar to a user u, then n is a neighbour of u [61].

In order to compute the nearest neighbours, the Euclidean distance can be used [64], which is computed as follows:

$$d(x, y) = \sqrt{\sum_{k=1}^{N}(x_k - y_k)^2} \tag{2.3}$$

A prediction for the user u for an item i is generated by analysing ratings for i from users in u's neighbourhood. A naive approach is to average all neighbours ratings for the item i [61]. In the formula below the rating of a neighbour n for an item i is represented by r_{ni} and the total number of neighbours which rated the item by N.

$$prediction(u, i) = \frac{\sum_{n=1}^{N} r_{ni}}{N} \tag{2.4}$$

This approach is considered naive, because it does not take into account that some neighbours might have higher similarity levels than others. More accurate predictions can be generated if the user similarities are taken into account. If $userSim(u, n)$ is a measure of the similarity between a target user u and a neighbour n, then a more accurate prediction can be computed as follows:

$$prediction(u, i) = \frac{\sum_{n=1}^{N} r_{ni} * userSim(u, n)}{\sum_{n=1}^{N} userSim(u, n)} \tag{2.5}$$

The above formula can also be extended by taking into account average adjusts for user's mean ratings and using the Pearson correlation [65]. One limitation of this approach is that if the data is sparse then the predictions might not be accurate enough. For example if there are only three common rated items, then the ratings could match almost exactly [61].

Herlocker et al., [66] have proven that accuracy can be improved by limiting the prediction to a users closest k-neighbours. To reduce processing time also sampling and clustering algorithms have been tried [61]. In sampling only a subset of users are selected and in clustering a user is compared to a group of users so that clusters of users similar to the target are quickly discovered and the nearest neighbours selected [61].

Item-based Nearest Neighbour Algorithm

The search for the nearest neighbour among a large user population can be a bottleneck [66]. Item based algorithms explore the relationship between items first and can be used to avoid this bottleneck [67]. This approach computes recommendations using similar items, that the user has liked. Sarwar et al., [67] stated that the relationships between items are static and that item-based algorithms might provide the same quality as user-based algorithms with less online computation.

In order to compute the similarity between two items i and j, the first step is to isolate the users which have rated both items [67]. One way to compute the similarity between two items is the cosine-based similarity method. In this method two items are thought of as two vectors and similarity is measured by computing the cosine of the angle between these two vectors [67]:

$$sim(i,j) = cos(\vec{i},\vec{j}) = \frac{\vec{i} * \vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2} \tag{2.6}$$

Other methods to compute the similarity are: the adjusted cosine similarity method and by using *Pearson-r* correlation [67].

Probabilistic Algorithms

In order to compute recommendations, probabilistic algorithms explicitly represent probability distributions [61]. A predicted rating r of a user u for an item i is calculated based on the most probable value or the expected value of r:

$$E(r|u,i) = \sum_r r * p(r|u,i) \tag{2.7}$$

Bayesian network models derive probabilistic dependencies among users or items [61]. Breese et al., [62] described a method for deriving and applying Bayesian networks using decision trees. For every item that can be recommended, a separate tree is constructed and the probabilities are displayed at the leaves of the tree [62].

Collaborative filtering systems are not always able to make personalized predictions [61]. When a user first registers with a collaborative filtering system, there are no ratings for this user, consequently personalized predictions cannot be computed (e.g., a neighbourhood of similar users cannot be established). This is known as a "cold start problem". The same situation can occur when a new item is registered. Because it has no ratings, no recommendations can be made. One way to overcome this problem is

to provide non-personalized recommendations (e.g., population averages or randomly selected items).

2.5.2 Content-based Recommender Systems

Content-based recommender systems are trying to recommend items to users, based on their preferences and similar liked items from the past [68]. The basic process performed by a content-based recommender system is matching up attributes of a user profile (in which the user's preferences and interests are stored) with attributes of items in order to find these recommendations.

One option is to let the user specify his or her own profile. This can be done by providing a simple graphical user interface with check-boxes representing certain attributes. This method however requires an effort from the user, which might be difficult to get or can become inaccurate if the user's preferences change over time [69].

Items are represented by a set of features called attributes or properties [68]. If each item is described by the same set of attributes, which can take only a certain set of values, then machine learning algorithms can be used to learn a user's profile [69].

Content-based recommender systems only recommend similar items to those already rated by users which leads to over-specialization and the system is not able to recommend items that are different, from anything that the user has seen before. This is also referred to as serendipity. One possible solution to address this problem is to recommend random items [68].

Content-based filtering and collaborative filtering can also be automatically combined. This is also known as a "hybrid approach". Such systems use content analysis to identify items and collaborative filtering to capture features like quality [61].

In this section we provided an overview of recommender systems together with a few basic methods that can be used for creating such systems and that can be applied for recommending data contracts in IoT dataspaces. Based on concluded data contracts and user ratings, recommendations can be made for the purchase of data produced by *Things*. These recommendations might also be applied to data contractual clauses, for example by making a recommendation for data produced by a *Thing* and also a price recommendation. However, in such cases a conflict of interest can appear: e.g., a provider would want to sell data at the highest possible price, whereas a consumer would want to purchase data at the lowest possible price.

2.6 State of the Art

In the past few years the number of IoT devices increased drastically. As mentioned before, it is estimated that the number of IoT devices, and therefore the amount of recorded data will continue to increase (due to different factors like connectivity price reduction) [5] [6] [7]. This has given rise to multiple IoT platforms and data markets, which facilitate purchasing, selling and exchanging data. In this section we are going to present current state of the art IoT data markets, their capabilities and their limitations.

2.6.1 MARSA

MARSA is a dynamic cloud-based marketplace for near real-time human-sensing data [9]. The platform distinguishes two types of data participants: data providers and data consumers and supports data delivery in near real-time from consumers to buyers [9]. In order to do so, the platform offers the possibility to transfer data streams directly between consumers and providers through an Application Programming Interface (API) (which can be used for machine-to-machine communication). MARSA also provides a service for data discovery, which allows users to search for data streams.

Furthermore, the platform supports multiple cost models (e.g., free usage, payment on time of subscription, etc.) and multiple contract models (e.g., obligation free contracts, customizable contracts) [9].

However, MARSA also has a few disadvantages. If a user wants to provide and buy data at the same time, two different accounts are necessary since the platform strictly distinguishes between these two types of users. Furthermore the platform is limited to the data discovery service and no data recommendation mechanism is provided, e.g. for making recommendations for *Things* which might be of interest to a user. The prototype also has the disadvantage that it was originally designed to provide near real-time data with the focus on machine-to-machine communication and no data contract negotiations are possible.

2.6.2 ThingStore

ThingStore is a platform for IoT application development and deployment [70]. This platform distinguishes between different categories of users and is designed to address the challenges of device heterogeneity, system complexity and use of network bandwidth.

ThingStore provides an IoT marketplace and APIs for smart service development and hosting and introduces an Event Query Language which can be used to query for events [70].

From an IoT data contract establishment perspective, ThingStore does not offer the possibility to specify or negotiate individual data contractual terms.

2.6.3 BDEX

BDEX [10] is a data exchange platform which enables buying and selling data in near real time. It allows the user to search for data based on age and quality and also to combine different data points. BDEX tries to quantify the data quality by introducing data scoring: every transaction, consumer and seller is scored and furthermore offers unbiased and impartial scoring. Another functionality offered by this platform is real-time monitoring of the data exchange, with the option to adjust the prices and to track the performance. Last but not least BDEX offers the possibility to target specific user groups.

At the time of the conducted online research BDEX did not offer the possibility of designing individual data contractual terms, e.g. it is not possible to sell the same data at the same time to different users, using a different price and differences in usage rights. Furthermore the data quality is limited to data scoring and does not offer further information by using data mining techniques, e.g. anomaly detection.

2.6.4 Microsoft Azure Marketplace

The Microsoft Azure Marketplace [11] offers a broad range of different types of data which can be accessed directly online or through an API. It supports data searching based on price, category and publisher. Regarding the cost model, only the following three pricing categories are supported: free data, free trials (which is usually limited to a few data transactions) and payment per data transaction.

From the point of view of data contracts, the Microsoft Azure Marketplace has a few disadvantages. It does not offer a dynamic cost model where the user pays based on the data size or can buy unlimited access for a limited time. Furthermore each publisher can specify the terms under which the data is provided. These terms apply to all users. This is a disadvantage, because it is not possible to negotiate these data contractual terms individually.

2.6.5 Further Platforms

Amazon Web Services Public Data Sets [71] provides public data for free, users have to pay only for data computation and data storage. The data is divided into different categories (e.g. astronomy, climate), however the platform provides no option for searching through the datasets, but just browsing through them. Furthermore no concrete information about the data quality is presented, just the text provided by the data publisher.

Tilepay [8] is an IoT micropayments platform that allows users to sell and exchange IoT data. It provides a user control mechanism, that allows users to specifically allow or deny data access and supports pseudonimity (i.e. disguised identity). However, at the time of the conducted research, the platform did not offer any information regarding the data contractual terms, nor if it is possible to monitor and recommend data contracts.

Thingspeak [72] is an IoT open data platform. It allows its users to access embedded devices and web services and to collect, store, and analyse data from sensors or actuators.

In order to do so, users are able to create a channel where the devices which contains multiple attributes like data fields, location fields and a status field. After a channel is created data can be written on that channel through provided APIs (e.g. Representational State Transfer (REST) services). Finally the data can be processed using MATLAB code. A major disadvantage of this platform is that users cannot exchange directly the data through channels and channels have to be made public first. Furthermore, at the time of the conducted research, the platform did not offer the possibility to establish individual data contracts.

CHAPTER 3

Requirements Analysis

In the previous chapter we elaborated upon IoT data contract establishment and provided an overview of current state of the art projects and data marketplaces. Based on the findings of the previous chapter, we are going extract a set of requirements in order to be able to implement a possible solution for the problem mentioned in Chapter 1. First we are going to provide a brief overview of the identified use cases, afterwards we are going to describe the functional requirements of these use cases and finally we are going to elaborate upon a few non-functional requirements.

3.1 Brief Use Case Description

Across IoT dataspaces many providers, having multiple *Things* can exist. Consequently, multiple potential customers can exist and some of these potential customers can also be *Thing* providers. Some of the current IoT platforms strictly distinguish between multiple types of users (see Chapter 2.6). However, in order to avoid the necessity of having multiple accounts we will assume that all actors of the system are users.

A user can either be a *Thing* provider (i.e., a user who owns a temperature sensor – a *Thing* – and wants to sell the streamed data produced by his/her *Thing*) or a *Thing* consumer (i.e., a user who wants to purchase data) or both at the same time. Furthermore, from a business perspective, a user is either a natural or a legal person. Each user can register only once and once registered, each user can act as a *Thing* provider and as a *Thing* consumer. For this reason, every user can register a *Thing*.

Furthermore it should be possible to bilaterally (see Figure 2.1) negotiate and conclude data contracts for the data produced by the previously registered *Things*. If the dataspace of the provider contains more than one *Thing*, a consumer has to be able to browse between the *Things* and select, and, negotiate data contracts for any number of *Things*.

After a data contract is concluded, it should be possible to monitor the data stream from a data quality perspective. The monitoring period of the data stream can be started/stopped by either one of the parties (i.e., *Thing* provider or *Thing* consumer). Furthermore, it should be possible to have multiple monitoring periods (e.g., if a data contract runs for 24 hours, it should be possible to have multiple monitoring periods of e.g., a few minutes/hours). After concluding a data contract, the *Things*, for which the data contract was concluded, can be rated.

Finally for each user a personal recommendation for a *Thing* can be computed, based on the users previous ratings and the ratings of his neighbourhood (see also use case illustration in Figure 3.1).

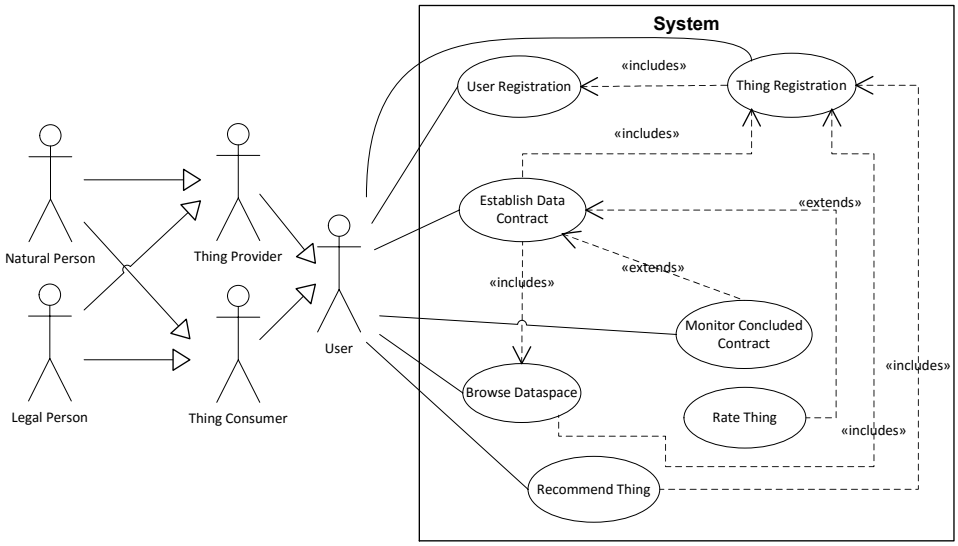


Figure 3.1: Use Case Diagramm

The previously mentioned use cases can be identified in several scenarios. For example let us consider a few providers, which are either natural persons (e.g., having a couple of sensors measuring the outside temperature), or companies (e.g., having multiple sensors measuring different environmental data). In this scenario different types of users might be interested in buying different types of data (e.g., local news agencies might be interested in weather data, universities and government in the measurements of air contaminants, etc.) and all stakeholders have one thing in common: they want to conclude data contracts and they wish to achieve it in the best possible way (e.g., purchase/sell at the lowest/highest possible price or access it for free).

One more scenario would be that of a smart city. For example in the Array of Things project, multiple *Things* are installed to monitor Chicago's environment [73]. In this

case different stakeholders might be interested in different types of data produced by different *Things*. For example local news agencies might be interested in temperature recordings, research institutes in air quality data, etc. Like in the previous scenario, all stakeholders have the same thing in common: they all want to access data and establish data contracts.

Another scenario, from an even larger perspective, has been identified by the Fraunhofer Institute. In this particular case an industrial dataspace was presented. This presented concept covers multiple branches with the goal of establishing a network of trusted data, which is produced by multiple sources: companies, *Things*, etc. [74]. Similar to the previous scenarios multiple stakeholders can be identified with the same goal: to establish data contracts for the purchase of data (e.g., automated measurements recorded by some robots in production lines) or access the data for free.

3.2 Functional Requirements

In the previous section we provided a short overview of the identified use cases. In this section we are going to present these use cases from a functional point of view, together with at least one representative scenario upon which the prototype will be evaluated (see Chapter 5.2) in order to demonstrate its utility.

3.2.1 User Registration

The user registration use case (see Table 3.1) addresses a limitation of currently existing platforms (see Chapter 2.6): a user should have to register only once, regardless of the role (i.e., provider or consumer) he/she is playing.

Use Case Element	**Description**
Name	User Registration
Primary Actor(s)	*Thing* provider – John, *Thing* consumer – Jane
Preconditions	None
Brief Description	Each user can create an account and afterwards use the system.
Postconditions	A user account is created and the user can log in.
Main Success Scenario 1	1. A natural person (John) opens the main page in a browser. 2. John selects the following type of person: 'natural person'. 3. John provides his first name and last name. 4. John provides his e-mail address and a password. 5. John provides his birth date. 6. John provides his home address. 7. John clicks on the 'Register' button.
Main Success Scenario 2	1. The employee (Jane) of a small company (Thing Inc.) opens the main page in a browser. 2. Jane selects the following type of person: 'legal person'. 3. Jane provides the company name and the company registration number. 4. Jane provides her e-mail address and a password. 5. Jane provides the address of the company. 6. Jane clicks on the 'Register' button.
Exception Flows	In case an e-mail address is invalid or already registered, an error message is received.

Table 3.1: Use Case 1 - User Registration

For the specification of a person, we chose a set of representative fields (see Table 3.1), serving as a proof of concept. When developing IoT data markets, it might in the interest

of multiple stakeholders to have more data about the person they are doing business with (e.g., for legal persons the beneficial owner).

3.2.2 *Thing* Registration

In order to conclude data contracts for data produced by a *Thing* (see description in Table 3.2) it should be possible to register a *Thing*. Furthermore every user has to be able to register a *Thing*. In this context a *Thing* is a device (of any kind, e.g., temperature sensor), which produces some kind of data.

Since the data, which is produced by *Things*, can have various formats, it is necessary to know the meta-model of the produced messages in order to be able to compute QoD and QoS metrics in further steps. For the sake of simplicity and as a proof of concept it should be possible for the provider to specify the meta-model of the produced messages of a *Things* upon its registration. Another option to retrieve the meta-model of a *Thing* would be to provide a custom component which can exchange information with other external systems that also manage *Things* and know their meta-model, such as [75].

At the end of Chapter 2.4 we presented one possibility to model data contracts. As mentioned in Chapter 1.2, we assumed that users are willing to negotiate data contracts for the purchase of data produced by registered *Things*. In order to ease the negotiation process, it should be possible for a *Thing* provider to specify a negotiation template. The negotiation template consists of sample data contractual clauses, which will be used in a later step for the negotiation process. As a proof of concept, from the proposed data contract model, all terms, except the QoD and the QoS, should be negotiable.

Use Case Element	**Description**
Name	*Thing* Registration
Primary Actor(s)	*Thing* provider
Preconditions	Registered and logged in user.
Brief Description	Each user can register a *Thing* in order to be able to conclude data contracts in a further step.
Postconditions	1. The *Thing* is saved, the user can edit its data and other users can find it in the dataspace of the provider. 2. The *Thing* provider receives notifications on how to configure his *Thing*, so that the platform can receive all *Things* messages.
Main Success Scenario	1. John logs in to his account. 2. John selects the option to register a *Thing*. 3. John provides the meta-model of the messages which are produced by his *Thing*. 4. John provides a resource ID for his *Thing*. 5. John provides a sample JSON object, similar to the ones which his *Thing* produces. 6. John provides a short description for the *Thing* (e.g., temperature sensor in the Vienna city center). 7. John provides information regarding the quality of data (i.e., completeness, conformity, accuracy, consistency and timeliness). 8. John provides information regarding the quality of service (i.e., device broadcasting frequency and the availability). 9. John specifies if his *Thing* should be monitored by the standard monitoring component of the system or not. 10. John specifies a pricing model (e.g., 10 EUR for 1 month of data) which will be used in the data contract template. 11. John specifies data rights clauses (i.e., derivation, collection, reproduction, commercial usage) which will be used in the data contract template. 12. John specifies purchasing policy clauses (i.e., contract termination, refunds, shipping) which will be used in the data contract template. 13. John specifies control and relationship clauses (i.e., warranty, indemnity, liability, jurisdiction) which will be used in the data contract template.
Exception Flows	If the *Thing* sample JSON, description or pricing model are not provided, an error message appears.

Table 3.2: Use Case 2 - *Thing* Registration

3.2.3 IoT Dataspace Browsing

As mentioned in Chapter 1 multiple *Things* can exist within an IoT dataspace. These *Things* can belong to one or to multiple providers. As a proof of concept we limit one IoT dataspace to only one provider (which in turn can own multiple *Things*).

However, we also assume that multiple dataspaces can exist, therefore it has to be possible for a user to browse or to switch between them and to see the *Things* that are bound to a dataspace (see description in Table 3.3).

Use Case Element	**Description**
Name	Browse Dataspace
Primary Actor(s)	*Thing* consumer
Preconditions	Registered and logged in user, at least on dataspace exists from another user with at least one registered *Thing*.
Brief Description	Each consumer is able to see all dataspaces and browse between them, so he can establish data contracts in further steps.
Postconditions	All *Things* within one dataspace are clearly visible.
Main Success Scenario	1. Jane logs in to her account. 2. Jane selects the option browse the dataspaces (from a menu or from an icon). 3. Jane sees a list of the available dataspaces and can select any of them and see all available *Things*.
Exception Flows	In case no dataspaces are available, a corresponding message is displayed.

Table 3.3: Use Case 3 - IoT Dataspace Browsing

3.2.4 Data Contract Negotiation

In order to support the bilateral negotiation of data contracts, according to the previously mentioned three phase model (locate potential business partners (1), negotiate an agreement (2) and conclude a contract (3) – see Section 2.2), the following data contractual clauses have been chosen to be negotiable: data rights, pricing, control and relationship and purchasing policy. The QoD and QoS clauses have been left out of the negotiation process because they could require a manual configuration for each concluded data contract for each *Thing*, or a more complex automatic configuration or alteration of the produced data, which would go beyond the scope of this thesis.

By negotiating the above mentioned data clauses, individual data contracts can be established (see Table 3.4).

Use Case Element	**Description**
Name	Negotiate Data Contract
Primary Actor(s)	*Thing* provider, *Thing* consumer
Preconditions	Registered and logged in user, at least on dataspace exists from another user with at least one registered *Thing*.
Brief Description	Individual data contracts can be concluded by negotiating data contractual terms.
Postconditions	A proposal for a data contract is always seen by the other party.
Main Success Scenario	1. Jane logs in to her account. 2. Jane selects the option browse the dataspaces (from a menu or from an icon). 3. Jane sees a list of the available dataspaces and selects Johns dataspace. 4. Jane selects one or more *Things* for which she wants to negotiate a data contract. 5.1. If only one *Thing* is selected, the contractual terms from the data contract template for that *Thing* is proposed to Jane. 5.2. If Jane selects multiple *Things*, no data contractual terms are pre-filled. 6. Jane sends the contract proposal to John. 7. John logs in to his account. 8. John sees Janes proposal and sends her a counter proposal. 9. Jane can see the counter proposal including the contractual terms of her last proposal. 10. Jane sends John a new counter proposal. 11. John accepts the received data contract proposal from Jane without modifying anything.
Exception Flows	In the pricing model, the price, the subscription period, the broker URL and the queue name (i.e., where exactly to send all *Thing* messages, once the contract is concluded) have to be specified, otherwise an error is displayed.

Table 3.4: Use Case 4 - Data Contract Negotiation

3.2.5 Data Contract Monitoring

The data contract monitoring (see Table 3.5) is bound to a monitoring period for one data contract and for all *Things*, for which the standard monitoring is enabled. As described in the previous use case (see Table 3.2), the standard monitoring flag is specified upon a *Things* registration and if it is enabled, then the data flow produced by this *Thing* can be monitored by the new proposed framework. After a data contract monitoring period is started (by any user), the monitoring should start automatically and when the monitoring period is ended (by any user), the monitoring should end automatically. Furthermore, during the subscription period of a data contract, it should be possible to have multiple monitoring periods.

Use Case Element	**Description**
Name	Monitor Concluded Data Contract
Primary Actor(s)	*Thing* provider, *Thing* consumer
Preconditions	Registered and logged in user, concluded data contract and subscription period of the concluded data contract has started.
Brief Description	For each *Thing*, which has the standard monitoring enabled, the monitoring can be started/stopped for a certain period for a data contract (e.g., simple start/stop button-function).
Postconditions	QoD and QoS metrics are available for every *Thing* within the data contract with the standard monitoring enabled. The monitored values refer only to the data contract subscription period.
Main Success Scenario	1. John logs in to his account. 2. John selects a concluded data contract, for which the subscription period is still running. 3. John can start/stop monitoring the data flow any time, which creates a monitoring period. 4. John can see the following QoS metrics for each monitoring period: number of samples, availability (in %), first message receival time, last message receival time. 5. John can see the following QoD metrics for each monitoring period: number of samples, message completeness (in %), message conformity (in %), average message age (in ms), average message currency (in ms). 6. If multiple monitoring periods exist, John can see them all.
Exception Flows	None.

Table 3.5: Use Case 5 - Concluded Data Contract Monitoring

3.2.6 *Thing* Rating

Once a data contract is concluded, a *Thing* consumer has to be able to submit a rating for each *Thing* which is bound to the data contract (see description in Table 3.6). This rating will be used in a later step for making recommendations.

Use Case Element	**Description**
Name	Rate Thing
Primary Actor(s)	*Thing* consumer
Preconditions	Registered and logged in user, concluded data contract and subscription period of the concluded data contract has started.
Brief Description	Each *Thing*, for which a data contract was concluded, can be rated once by the consumer.
Postconditions	The rating is persisted into the database.
Main Success Scenario	1. Jane logs in to her account. 2. Jane selects a concluded data contract. 3. Jane submits a rating for each *Thing* from the data contract on a scale from 1 to 5 stars.
Exception Flows	In case a *Thing* has already been rated by the user, a message is displayed, among with the submitted rating value.

Table 3.6: Use Case 6 - *Thing* Rating

3.2.7 Data Contract Recommendation

In order to stimulate data contract conclusions it has to be possible to make data contract recommendations. The recommendations are based on the submitted ratings of other users (see scenarios from Table 3.7).

In order to avoid a conflict of interest between *Thing* providers and *Thing* consumers (e.g., a provider would want to sell data at the highest possible price, whereas a consumer would want to purchase data at the lowest possible price), the recommendation is only bound to recommending *Things* and not to recommending data contractual clauses.

Use Case Element	**Description**
Name	Recommend Data Contract
Primary Actor(s)	*Thing* provider, *Thing* consumer
Preconditions	Registered and logged in user
Brief Description	Each user receives a personal recommendation based on previously concluded data contracts and the neighbourhood of the user.
Postconditions	A recommendation or a message that a recommendation cannot be made is clearly visible.
Main Success Scenario 1	1. Jane logs in to her account. 2. Jane has no previously concluded data contracts and in the platform multiple *Things* exist, which have been rated by other users. 3. Jane receives a recommendation for the highest rated *Thing*.
Main Success Scenario 2	1. Jane logs in to her account. 2. Jane has no previously concluded data contracts and in the platform no rated *Things* exist. 3. Jane receives a recommendation for a randomly chosen *Thing*.
Main Success Scenario 3	1. Jane logs in to her account. 2. Jane has no previously concluded data contracts and in the platform no *Things* exist. 3. Jane receives no recommendation, just a notification, that no recommendation can be made.
Main Success Scenario 4	1. Jane logs in to her account. 2. Jane has previously concluded data contracts and in the platform multiple rated *Things* from multiple users exist. 3. Jane receives a recommendation for a *Thing* she didn't conclude a data contract yet and has the highest rating in the nearest neighbourhood, which is computed using the Euclidean distance (see Formula 2.3) between commonly rated *Things*.
Exception Flows	None.

Table 3.7: Use Case 7 - Recommend Data Contract

3.3 Non-Functional Requirements

In the previous section functional requirements, that need to be fulfilled, have been described. In this section a few identified non-functional requirements, which apply for the new proposed framework, are presented:

- The developed platform has to be able to deal with a high number of users, *Things* and data contracts, thus it has to be scalable.
- The developed architecture should be based on small services (e.g., microservices) which can run independently.
- The developed architecture should be extensible by custom components.

CHAPTER 4

Contract-aware Framework

Based on the conducted literature research from Chapter 2 and the defined requirements from Chapter 3 an extensible contract-aware framework was designed, which can serve as a possible solution for the problem mentioned in the introduction of this thesis (see Chapter 1). In this Chapter we are going to present this framework: at the beginning we are going to provide a short overview from an architectural point of view and afterwards we are going to describe each of the frameworks components in more detail.

4.1 Architecture

In order to support all required features and to achieve a better separation of concerns, the architecture of the framework was developed based on microservices. This way small services can manage different aspects. The following types of services have been identified: *DataContract*, *Monitoring*, *Recommending* and further custom plugins/services (see Figure 4.1).

Each service can be scaled individually and can communicate with a database on its own. All services are managed through a *ServiceHandler*. The main purpose of the *ServiceHandler* is to integrate multiple services of different types including custom made, which makes the architecture easily extensible by custom components. For example *Thing* providers can create a custom made microservices for other activities (e.g., custom monitoring or further features) and register them via the API of the *ServiceHandler*.

In the new proposed framework, it has been assumed that *Things* publish data to IoT data hubs. Since one single IoT device can produce a lot of data (e.g., a temperature sensor can record the temperature every hour, minute or every second), and this data can be stored and/or published in multiple locations, it has been further assumed, that *Things* publish the produced data as messages to message brokers.

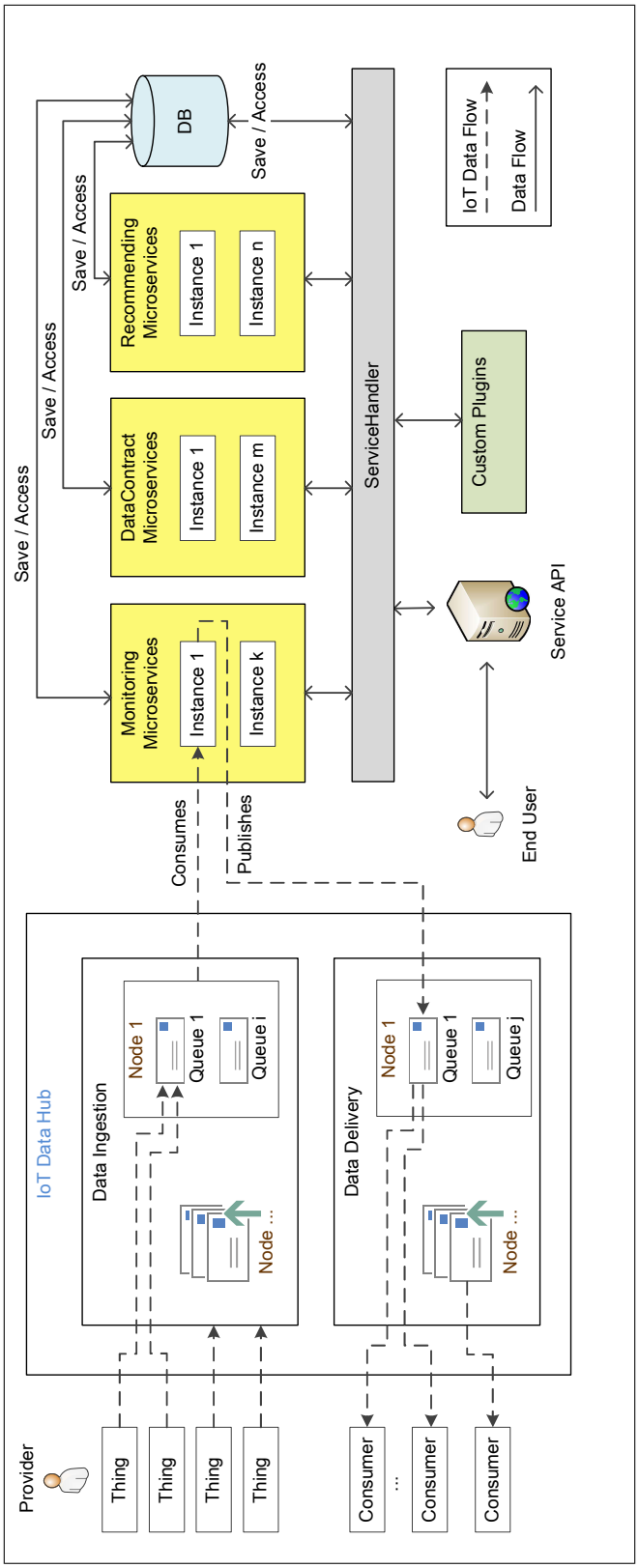


Figure 4.1: Framework Architecture

We assumed that *Things* publish the produced data to data ingestion nodes (see Figure 4.1). One node can consist of one or more message brokers/queues. Another assumption that has been made, is that an IoT provider is willing to configure his/her *Thing* to send all messages to a specific queue and that messages might also contain specific custom parameters/properties (e.g., queue topic or custom message property). By doing so, each consumed message can be immediately mapped to the *Thing* which produced the respective message. For example, let us consider two *Things* t_1 and t_2 that publish messages to the same queue. By attaching a custom property to every message, e.g., $thingid = t_1$ resp. $thingid = t_2$, the producing *Thing* can always be identified.

Finally, we made the assumption, that all messages are string representations of JSON objects [1]. However, it is not assumed, that the produced JSON objects have a common model or that the model of the messages is known. The model of a message is only known, if the meta-model of the *Thing* (which produced the message) is known. As described in use case 2 (see Table 3.2) the meta-model of a *Thing* can be provided upon its registration.

The overall functionality of the framework can be accessed through a provided API of the *ServiceHandler*. Alternatively a GUI server can offer end-user access, which in turn accesses the same *ServiceHandler* API.

4.2 Data Contract Management

The data contract service is responsible for managing data contracts and consists of a layered architecture with the following layers: (1) Data Access Object (DAO) layer, (2) a business layer and (3) a communication/API layer. All operations regarding data contracts are run through this service, including the registration of *Things*, for which the data contracts are established for. *Things* and data contracts can be updated and created by accessing the provided API.

In order to be able to support individual data contract establishment, this microservice consists of two components: a *Thing* component and a data contract component. The *Thing* component provides all necessary services for the *Thing* management and the data contract component provides all necessary services for the establishment and maintenance of data contracts (an example of a data contract is provided Listing 4.1).

As we can see in the nested object *dataContractMetaInfo* the most important data contract information is saved (i.e., contract ID, creation date, etc.). The rest of the contract consists of the agreed data contractual clauses, the IDs of the *Things* from which the data shall be broadcasted and a flag indicating that the monitoring is currently enabled for this data contract. The data contractual clauses have been chosen based on the conducted literature research and based on the extracted requirements (see Chapters 2.4 and 3.2).

[1] Having to deal with other types of messages (e.g., XML) could be an option and is highly possible, but would go beyond the scope of this thesis

Listing 4.1: Data Contract Example

```
{
 "dataContractMetaInfo": {
  "contractId": "c123",
  "creationDate": "2016-10-06 00:11:03",
  "active": true,
  "party1Accepted": true,
  "party2Accepted": true,
  "revision": 5,
  "party1Id": "p123",
  "party2Id": "p456",
 },
 "dataRights": {
  "derivation": false,
  "collection": true,
  "reproduction": false,
  "commercialUsage": true
 },
 "pricingModel": {
  "price": 3,
  "currency": "EUR",
  "transaction": false,
  "numberOfTransactions": 0,
  "startDate": "2016-10-07 00:00:00",
  "endDate": "2016-10-08 00:00:00",
  "brokerURL": "tcp://127.0.0.1:61616",
  "queueName": "consumerQueue"
 },
 "controlAndRelationship": {
  "warranty": "None",
  "indemnity": "None",
  "liability": "None",
  "juristiction": "AustriaVienna"
 },
 "purchasingPolicy": {
  "contractTermination": "Automatic",
  "shipping": "Automatic",
  "refund": "None"
 },
 "thingIds" : [ { "thingId" : "..." }, ... ],
 "monitoring": true
}
```

Each data contract (see class diagram in Figure 4.2) is concluded between two parties, which can be either natural or legal persons. Furthermore a data contract is concluded for a list of *Things* and contains certain clauses (i.e., pricing model, control and relationship, data rights, purchasing policy). Upon every *Thing* registration, data contractual clauses have to be specified. These clauses are automatically used to set up a data contract template which is used for the negotiation.

A *Thing* contains a list of ratings, QoD and QoS information and the messages, that *Things* produce, are specified by a meta-model. The meta-model is represented by a list of attributes. Each attribute has a certain data type (e.g., STRING, BOOLEAN, DATE, etc.), a few optional properties (e.g., an identifier flag, which can be set, if the attribute acts as an identifier) and, if the attributes data type is not primitive, the attribute can also have a meta-model per se. When data contracts are negotiated, a data contract trail is saved. The data contract trail represents the negotiation history, containing all values from all negotiated clauses.

When data contracts are negotiated, a data contract trail is automatically saved. The data contract trail contains the negotiation history as data contract trail entries, with

all values from all negotiated clauses. For each data contract offer or counter-offer a trail entry is saved. An example of a data contract trail is provided in Listing 4.2. In this example we can see that two offers have been exchanged and two data contract trail entries exist, with different values at various clauses (e.g., data rights derivation, reproduction and pricing).

By modelling only two parties in a data contract, the negotiation process is limited to bilateral negotiations. Based on the scenarios mentioned in the previous section, other types of negotiation are also suitable. For example in an industrial dataspace multi-bilateral or nested negotiations might also be considered. However, as a proof of concept, we modelled our framework (and later on the prototype) to be able to deal with bilateral negotiations only, as supporting the other types would go beyond the scope of this thesis.

Listing 4.2: Data Contract Trail Example

```
{
 "contractId": "c123",
 "revision": 1,
 "clausesTrail": [{
  "dataRights": {
   "derivation": true,
   "collection": true,
   "reproduction": true,
   "commercialUsage": true
  },
  "pricingModel": {
   "price": 3.0,
   "currency": "EUR",
   "transaction": false,
   "numberOfTransactions": 0,
   "subscription": {
    "startDate": "2016-10-01 00:00:00",
    "endDate": "2016-10-31 23:59:59",
    "brokerURL": "tcp://127.0.0.1:61616",
    "queueName": "customQueue"
   }
  },
  "purchasingPolicy": {
   "contractTermination": "Automatic",
   "shipping": "Automatic",
   "refund": "Full"
  },
  "controlAndRelationship": {
   "warranty": "2 years",
   "indemnity": "None",
   "liability": "None",
   "juristiction": "Austria/Graz"
  }
 },
 {
  "dataRights": {
   "derivation": false,
   "collection": true,
   "reproduction": false,
   "commercialUsage": true
  },
  "pricingModel": {
   "price": 10.0,
   "currency": "EUR",
   "transaction": false,
   "numberOfTransactions": 0,
   "subscription": {
    "startDate": "2016-10-01 00:00:00",
    "endDate": "2016-10-31 23:59:59",
    "brokerURL": "tcp://127.0.0.1:61616",
    "queueName": "customQueue"
   }
  },
  "purchasingPolicy": {
   "contractTermination": "Automatic",
   "shipping": "Automatic",
   "refund": "None"
  },
  "controlAndRelationship": {
   "warranty": "None",
   "indemnity": "None",
   "liability": "None",
   "juristiction": "Austria/Vienna"
  }
 }]
}
```

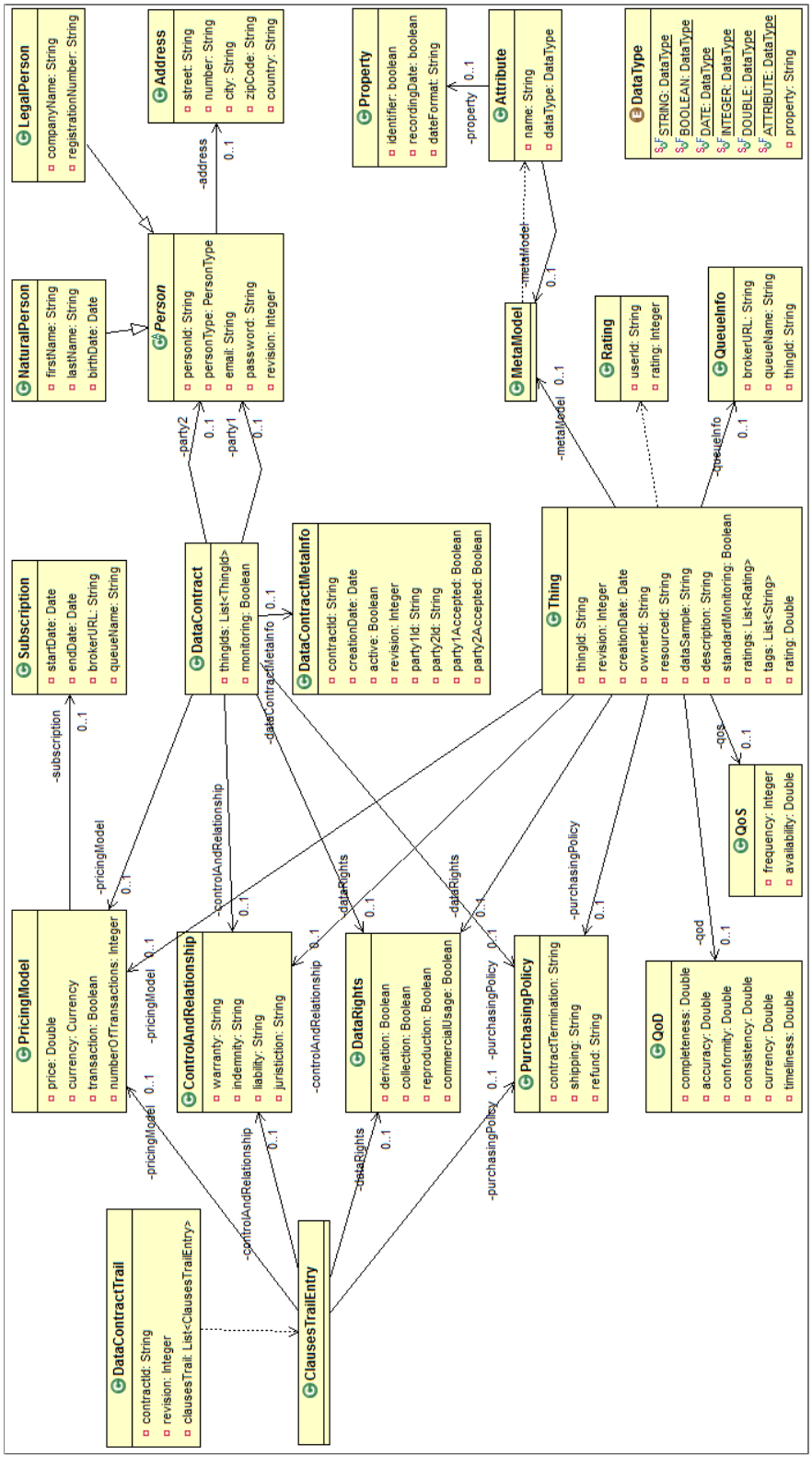


Figure 4.2: Data Contract Data Model

4.3 Data Contract Monitoring

Within an IoT dataspace, lots of devices can exist, which produce different types of data. This data is published at different frequencies and has to be consumed and redirected to all relevant subscribers (see Figure 4.1).

The *Monitoring* microservice is responsible for the monitoring and redistribution of all incoming messages from all devices and consists of a similar layered architecture as the data contract management service.

In this context two main challenges have been identified:

1. Data redistribution: x data contracts can be concluded, each for data produced by 1 to y *Things*.

2. Data monitoring: x *Things* can produce $z \leq x$ different types of data at $t \leq x$ different frequencies.

In order to solve the above mentioned challenges we propose a new concept of operators called *dataspace operators*. We define dataspace operators *op* to be operations which are applied to the consumed data from an IoT data hub (see Figure 4.1). For the previously mentioned challenges two operator types can be used: data operators op_D and monitoring operators op_M.

Data operators can be used to apply all necessary operations so that the consumed data is distributed accordingly to concluded data contracts. As a concept, depending on the situation multiple operators can be applied. For the concrete data redistribution problem, the appliance of a filtering operation will suffice in most cases (i.e., as long as data tampering is not necessary). If the consumed data has to be altered before it is redistributed to the consumers, then further operators could be used (e.g., split, merge, etc.).

The monitoring operators can be used for the quality assurance monitoring. For this purpose we propose the usage of two monitoring operators: op_{m1} for the QoD monitoring and op_{m2} for the QoS monitoring.

Algorithm 4.1 shows how the proposed dataspace operators can be applied on a data stream for the QoD and QoS computation. One *Monitoring* microservice consumes messages from one IoT data hub node, where multiple *Things* publish their messages. For each registered *Thing* the filtering data operator op_D is applied. The purpose of this filtering operation is to retrieve a list of all valid data contracts which have been concluded and include data produced by the current *Thing*.

Based on a configurable amount of consumers, multiple messages can be read at the same time. Each message is delivered to all relevant subscribers (i.e., consumers) according to the concluded data contracts.

After the message delivery the second type of monitoring operators can be applied in order to compute QoD and QoS metrics (see Algorithms 4.2 and 4.3).

Algorithm 4.1: Data contracts monitoring algorithm

Input: thingsList, nrOfConsumers, allDataContracts
Output: void
LOOP thingsList
1: **for** *thing* in thingsList **do**
2: $dataContractList = op_{d1}(thing)$
LOOP nrOfConsumers
3: **for** *consumer* in nrOfConsumers **do**
4: start *consumer* and consume *message* from *thing.queue*
5: $messageReceivalTime := currentTime$;
6: deliver *message* to subscriber
7: **if** ($currentTime$ in $sampleMonitoringTime$) **then**
8: $messageDeliveryTime := currentTime$;
9: $op_{m1}(thing.metaModel, message)$;
10: $op_{m2}(thing, messageReceivalTime)$;
11: **end if**
12: **end for**
13: **end for**
14: **return**

One possible solution for the computation of the QoD is presented in Algorithm 4.2. In this case the computation is based on the meta-model of the produced messages of a *Thing*. If the meta-model is know, then a message can be validated against it. If all attributes of a message are existent, then the message is considered to be complete. Furthermore, if the data types of the properties of a message match the expected data type (as specified by the meta-model), then the message is considered complete (also see Chapter 2). All attributes of a message are traversed and checked against completeness and conformity. If the data type of one attribute is not primitive (e.g., boolean, integer), then the function recalls itself.

One possible solution for the computation of the availability is illustrated in Algorithm 4.3. Here f_e is the expected device broadcasting frequency, t_{fmr} is the first message receival time, t_{mr} is the current message receival time, m_e represents the number of expected messages and m_t is the number of total received messages. If there are no current measurements of the availability, then the current received message is the first and only message so the availability is set to 100%.

Algorithm 4.2: QoD computation algorithm - op_{m1}

Input: $metaModel$, $message$
Output: $completeness$, $conformity$

$completeness$:= true; $conformity$:= true;
{loop all metaModel attributes}

```
1: for attribute in metaModel do
2:   if (message not contains attribute.name) then
3:     completeness := false;
4:     continue;
5:   else
6:     currentAttr := message.attribute
7:     if (currentAttr.dataType != attribute.dataType) then
8:       conformity:= false;
9:     end if
10:  end if
11:  if (attribute.dataType is not primitive Type) then
12:    traverseMetaModel(attribute.metaModel, message.attribute);
13:  end if
14: end for
15: return
```

Algorithm 4.3: QoS computation algorithm - op_{m2}

Input: $thing$, t_{mr}
Output: $thing.monitoredqos$

availability := 0;

```
1: f_e := thing.f_e;
2: if (thing.monitoredqos != NULL) then
3:   t_fmr := thing.monitoredqos.t_fmr;
4:   Δt := t_mr - t_fmr;
5:   m_e := Δt / f_e;
6:   m_e++;
7:   thing.monitoredqos.m_e := m_e;
8:   thing.monitoredqos.m_t++;
9:   thing.monitoredqos.availability := m_t / m_e;
10: else
11:   thing.monitoredqos.t_fmr := t_mr;
12:   thing.monitoredqos.m_e := 1;
13:   thing.monitoredqos.m_t := 1;
14:   thing.monitoredqos.availability := 1;
15: end if
16: return
```

However, if previous measurements exist, then based on the device broadcasting frequency, the receival time of the first message and the receival time of the last message, the number of expected messages can be computed. Let us assume a device which is broadcasting data every second and the monitoring is enabled for the following period of time: starting at "12:00:00" until "12:00:59". In this particular case we expect one message at "12:00:00", one at "12:00:01", etc. until "12:00:59" totalling 60 messages, or $m_e = 60$. With every received message m_t is increased by 1. If at "12:00:59" our device broadcasted as expected, then $m_t = 60$, which leads to an availability of 100%. However, if the device broadcasts data every 2 seconds (e.g., due to misconfiguration), then $m_t = 30$, which leads to an availability of 50%.

4.4 Recommending Management

Depending on the scenario, within an IoT dataspace, thousands of devices may exist, which are all producing data. In this particular case it may be useful to have a system which makes recommendations, that aligns with a users interests.

Algorithm 4.4 shows a possible hybrid approach that can be used, to recommend data produced by *Things*, that a current user did not buy, but his/her neighbours did. First the Euclidean neighbourhood is computed using the Euclidean distance (see Chapter 2.5). If the neighbourhood exists, the neighbours are sorted based on their distance, with the nearest distance first. If at least one neighbour has concluded a data contract for one *Thing* that the current user did not conclude a data contract, then this *Thing* is returned. If no *Thing* can be found in the neighbourhood, or if no such neighbourhood exists, the top rated *Thing* is returned. In case there are no ratings, and the top rated *Thing* does not exist, then a random *Thing* is proposed. Finally, if no *Things* are existent, then a random *Thing* cannot be proposed and a recommendation cannot be made.

Algorithm 4.4: *Thing* recommendation algorithm

Input: *userId*
Output: *thing*

```
1: Map<String, Double> map = computeEuclideanNeighborhood(userId);
   {Map<userId, distance>}
2: if (map is not empty) then
3:    Map<String, Double> sortedMap = sortMapByNearestDistanceFirst(map);
4:    for entry < String, Double > in sortedMap do
5:       thing := findThingNeighbourHasButUserHasNot(entry.String, userId);
6:       if (thing is not NULL) then
7:          return thing;
8:       end if
9:    end for
10: end if
11: thing := computeTopRatedThing(); {Thing with highest average rating}
12: if (thing is not NULL) then
13:    return thing;
14: else
15:    thing := getRandomThing();
16:    if (thing is not NULL) then
17:       return thing;
18:    end if
19: end if
20: return NULL;
```

Algorithm 4.4 recommends *Things* and not data contractual clauses because a conflict of interest can exist between *Thing* providers and *Thing* consumers (e.g., a provider

would want to sell data at the highest possible price, whereas a consumer would want to purchase data at the lowest possible price).

One disadvantage of Algorithm 4.4 is that it does not take external events into consideration. Let us assume multiple regions where multiple *Things* measure environmental data. If an important environmental event occurs in one single region (e.g., storm, hurricane, earthquake, etc.), then the data produced by the *Things* of that region might gain more importance than the data produced by the *Things* of other regions.

One option to deal with such scenarios is presented in Algorithm 4.5. This algorithm is very similar to the previous one with two main differences: (1) the neighbourhood is left out and (2) the recommendation is based on tags.

In this case we assumed, that *Things* can have multiple Tags (e.g., for a temperature sensor *#Vienna, #Austria, #Temperature*). If an important event occurs (e.g., thunderstorm in Vienna), *Thing* recommendations can be made based on these tags (e.g., finding *Things* tagged with *#Vienna*).

Algorithm 4.5: *Thing* tag-based recommendation algorithm

Input: *tag*
Output: *thing*

```
1: thing := computeTopRatedThing(tag);
2: if (thing is not NULL) then
3:    return thing;
4: else
5:    thing := getRandomThing(tag);
6:    if (thing is not NULL) then
7:       return thing;
8:    end if
9: end if
10: return NULL;
```

CHAPTER 5

Experiments

In the previous Chapter we presented the design of a new extensible contract-aware framework. Based on this design and the requirements from Chapter 3, a prototype was build to serve as a possible solution to the problem description mentioned in Chapter 1. In this Chapter we are going to present the implemented prototype [1] and evaluate it from two perspectives: from end user perspective and from a technical perspective.

5.1 Prototype

Based on the previously presented architecture (see Figure 4.1), we implemented a data marketplace prototype – called *IDAC*. *IDAC* was implemented as a proof of concept, can run on its own, can be plugged in to existing IoT data hubs and also act as an IoT data hub.

5.1.1 Implementation

We implemented *IDAC* in the Java programming language (version 1.8). As shown in Figure 4.1, *IDAC* consists of several components and was designed as an extensible and scalable distributed system, which can run on different physical machines.

As a database the non SQL (NoSQL) database MongoDB was used. MongoDB was chosen because it is schema less and due to its performance and scalability options [76].

As mentioned in Chapter 4.1, for the sake of simplicity, we assumed that *Things* publish data to message brokers/queues. As a proof of concept we simulated an IoT data hub ingestion node and a delivery node (see Figure 4.1), by using the Apache ActiveMQ [77] message broker.

[1] Complete code available at: `https://github.com/e0725439/idac`, accessed November 13, 2016

For the implementation of all services, the Apache Camel framework [78] was used. The Apache Camel framework was chosen, due to the possibility to use multiple enterprise integration patterns and to integrate different technologies at the same time. In order to run each service, the integrated Jetty server, that comes with Apache Camel framework, was used. The Graphical User Interface (GUI) was developed using PrimeFaces [79] and run on an Apache Tomcat [80].

Furthermore, all services consist of multiple components. Each component provides REST services, so that the components functionality can be accessed. Furthermore, the *ServiceHandler* and the microservices also communicate via REST service calls. This gives the possibility to run each service on entirely different physical machines.

For the sake of completeness, a simple balancing mechanism was implemented for the monitoring services, which balances the load equally between all service instances, based on the service type. For example if two *DataContract* microservices ms_1 and ms_2 are registered and two external requests r_1 and r_2 are received to establish the location of a *DataContract* service, then one request receives the location of one *DataContract* microservice and the other request receives the location of the other microservice, e.g., $r_1 \rightarrow ms_1$ and $r_2 \rightarrow ms_2$. Beyond the previously mentioned feature, the *ServiceHandler* assigns *Things* publishing information to each registered monitoring microservice instance. This way one monitoring microservice instance is responsible for consuming messages published by multiple *Things* from one data ingestion node only (see Figure 4.1).

In order to achieve a better separation of concerns and to avoid code repeatability, *IDAC* consists of several projects (e.g., common model project, common test project, etc.). Each project can be build using the build automation tool Maven [81]. All build configurations and dependencies to other projects are managed in *pom.xml* files. Each project was created using the Maven standard directory layout:

- *src/main/java*: Java classes.
- *src/main/resources*: Configuration files.
- *src/test/java*: JUnit tests.

IDAC was implemented using test-driven development, summing in total 114 JUnit tests. All tests run automatically during each build. The prototype was configured as a multi-module project, with the parent on the highest directory level. Running a build (e.g., *mvn clean install*) on this level ensures the correct build order of all artefacts.

5.1.2 Configuration and Deployment

As mentioned in the previous section, all projects use the Maven standard directory layout. All necessary configurations can be found in the directory *src/main/resources*, where the following properties files exist:

- *mongo_db.properties*: Database configuration
- *webservices.properties*: Jetty server configuration
- *activemq.properties*: ActiveMQ consumer and broker Uniform Resource Locator (URL) configuration (monitoring services only)

After a build job is executed successfully, runnable JAR files are created in each project. For the deployment of the prototype, the following runnable JAR files are needed, in each case together with their corresponding *lib* folder (which contains all necessary dependencies):

- ac.at.tuwien.mt.servicehandler-1.0.0-SNAPSHOT.jar
- ac.at.tuwien.mt.datacontract-1.0.0-SNAPSHOT.jar
- ac.at.tuwien.mt.monitoring-1.0.0-SNAPSHOT.jar
- ac.at.tuwien.mt.recommending-1.0.0-SNAPSHOT.jar

After all services have been started (e.g., process started using *java -jar ...*), all microservices have to register to the ServiceHandler. In order to do so, all microservices provide a GET REST service, which automatically registers the service to the ServiceHandler. Furthermore a shut down REST service is provided, which attempts a graceful shut down of the respective service (see Table 5.1).

Type	**Path**	**Service Description**	**Expected Response**
GET	/ping	Verifies service availability.	204 No Content.
GET	/register	Registers service to ServiceHandler.	200 OK.
GET	/shutdown	Attempts a graceful shut down.	204 No Content.

Table 5.1: Managing Services via REST API

5.2 Evaluation

In the previous section we elaborated upon some of the prototypes implementation details. In this section we are going to evaluate the prototype using two evaluation methods. The first evaluation method is from an end user perspective and is a descriptive evaluation, based on the representative scenarios from the previously described use cases, to demonstrate the utility of the prototype. The second evaluation method is from a technical perspective and is analytical, more specific a dynamic analysis with the focus on performance.

5.2.1 Evaluation Environment

The prototype was evaluated in two different environments, with the properties shown in Tables 5.2 and 5.3. These system settings were chosen to demonstrate the performance and the reliability of the prototype, even when running within a limited environment.

Property	**Value**
Operating System	CentOS Linux 7.2.1511
Kernel	64-bit
CPU	Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz
CPU Cores	2
Memory	8 GB

Table 5.2: Evaluation Environment 1 - Virtual Server

Property	**Value**
Operating System	Windows 10
Kernel	64-bit
CPU	Intel(R) Core(TM) i3 M370 @ 2.40GHz
CPU Cores	2
Memory	8 GB

Table 5.3: Evaluation Environment 2 - Physical Machine

5.2.2 Descriptive Evaluation

In this section we will analyse the prototype from an end user perspective to demonstrate its utility. This evaluation is based on the representative scenarios described in the use cases in Chapter 3.2 and was conducted completely in environment 1 (see Table 5.2).

At the beginning, the user registration was evaluated based on the described main success scenarios from Use Case 1 provided in Table 3.1. In this case two person registrations were completed successfully: a natural person (John Doe) and a legal person (Jane Doe). These two registered persons were further used in all other use case evaluations.

We further evaluated the *Thing* registration based on the described main success scenario from Use Case 2 provided in Table 3.2. The registration of the following three types of *Things* was conducted:

1. Real device: Temperature sensor DS18B20 connected to Raspberry Pi (see message example in Listing 5.1).
2. Simulated device: Water quality sampling data downloaded from [82].
3. Simulated device: Mobile device measurements obtained with [83].

Listing 5.1: *Thing* Message JSON Object Example

```
{
        "thingId": "42921004292100",
        "temperature" : "28.0",
        "scale" : "Celsius",
        "time" : "2016-10-31'T'15:42:59.00.123+0200"
}
```

For each type of *Thing* two devices were registered. In order to be able to simulate the dataspace browsing Use Case 3 (see Table 3.3) one further person was registered and that person registered one *Thing*. Afterwards, the previously registered user (i.e., Jane Doe) was able to see two dataspaces and the registered *Things* from each dataspace.

For the previously registered devices (assuming their IDs are: T1, T2, W1, W2, M1, M2), several data contracts were established by negotiation (see Table 5.4), each consisting of different *Things*. In all cases two negotiation steps were simulated. First, the *Things* were selected for which a data contract should be established (see Figure 5.1). Afterwards, a data contract offer was sent to the provider. The provider responded with a counter offer, which the consumer accepted. Each sent offer, represents a data contract trail and was saved into the database. The other party was always able to clearly see the differences between the received data contractual clauses and the sent data contractual clauses. For example, in Figure 5.2 a data contract offer was sent, in which all data rights values were enabled, however, the provider sent a counter-offer, where only the commercial usage of the data was allowed. Another example of a counter-offer is provided in Figure 5.3, where a price difference can be clearly seen.

Data Contract	*Things*
DC1	T1
DC2	W1
DC3	M1
DC4	T2, W2
DC5	T2, M2
DC6	W2, M2
DC7	T1, W1, M1

Table 5.4: Negotiated Data Contracts

After the successful establishment of the data contracts, each data contract was monitored for different monitoring periods. For example in Figure 5.4 a data contract consisting of one *Thing* was monitored for a period of approx. 1 hour, time during which 425 messages were successfully analysed. An example of a data contract monitoring consisting of two *Things* can be seen in Figure 5.6.

One *Thing* was configured to broadcast messages at a lower frequency than the expected one. The expected frequency was 10 seconds, but the actual configured frequency was 11 seconds. The data contract, which consisted of this *Thing* was monitored twice. The first monitoring period lasted approx. 1 minute, time during which such an anomaly could easily be overlooked. However, a second longer monitoring period, showed decrease of the device's availability (see Figure 5.5).

The *Thing* recommendation was evaluated based on the described main scenarios from the Use Cases 6 and 7 (see Tables 3.6 and 3.7). As mentioned previously, in order to avoid a conflict of interest between providers and consumers, the recommendations were made for *Things* only. In all cases where recommendations could be made, the user was redirected to a similar page as the one shown in Figure 5.1, however, consisting only of one *Thing*.

The successful completion of all previously mentioned experiments demonstrates that in *IDAC* different *Things* (i.e., temperature sensor and simulated devices with data from [82] and [83]), which produce different types of data, can be combined together in data contracts as shown in Figure 1.1, which illustrates the concept of this new framework.

Furthermore, the data contracts can be bilaterally and individually negotiated according to the five stage negotiation systems model (see Chapter 2.2). After a successful data contract establishment, the data stream can be successfully monitored, without an explicit time period limitation, while the data contract subscription period is still running. Finally we have shown that recommendations for *Things* can be made based on user ratings. As a conclusion, it is safe to assume that this prototype can serve as a possible solution for the problem described in Chapter 1.

Figure 5.1: *Thing* Selection Screenshot

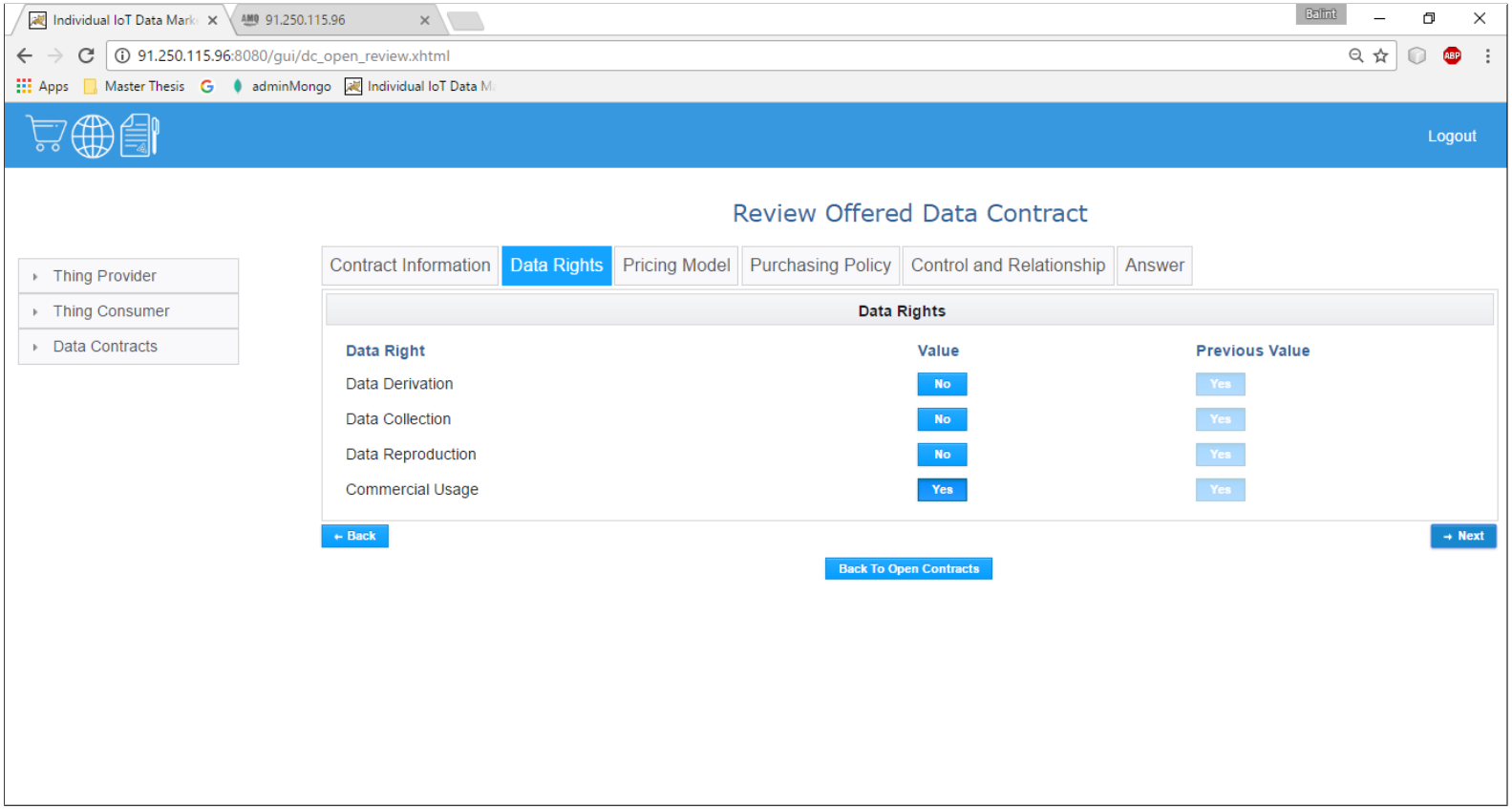


Figure 5.2: Negotiation Screenshot 1

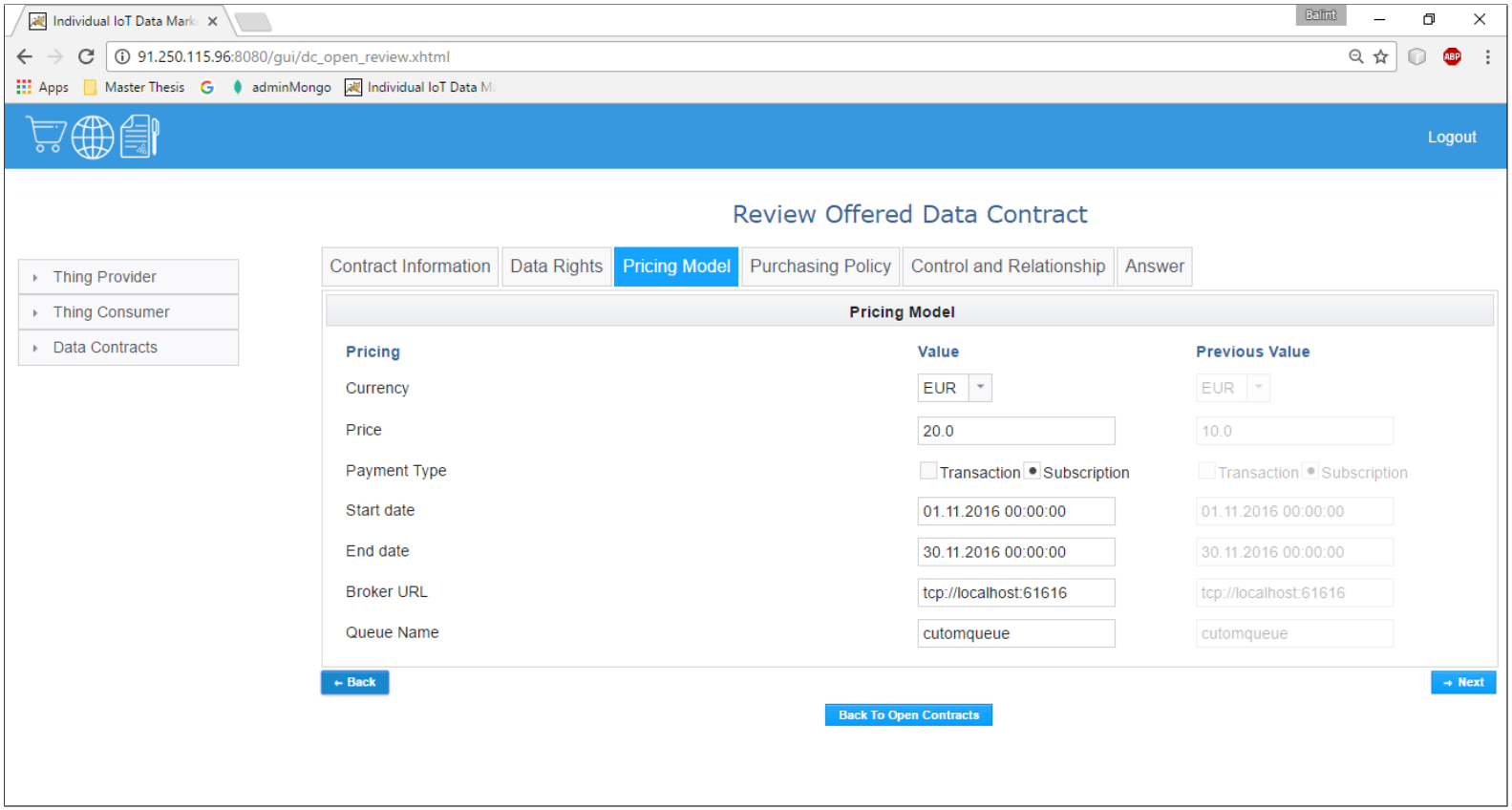


Figure 5.3: Negotiation Screenshot 2

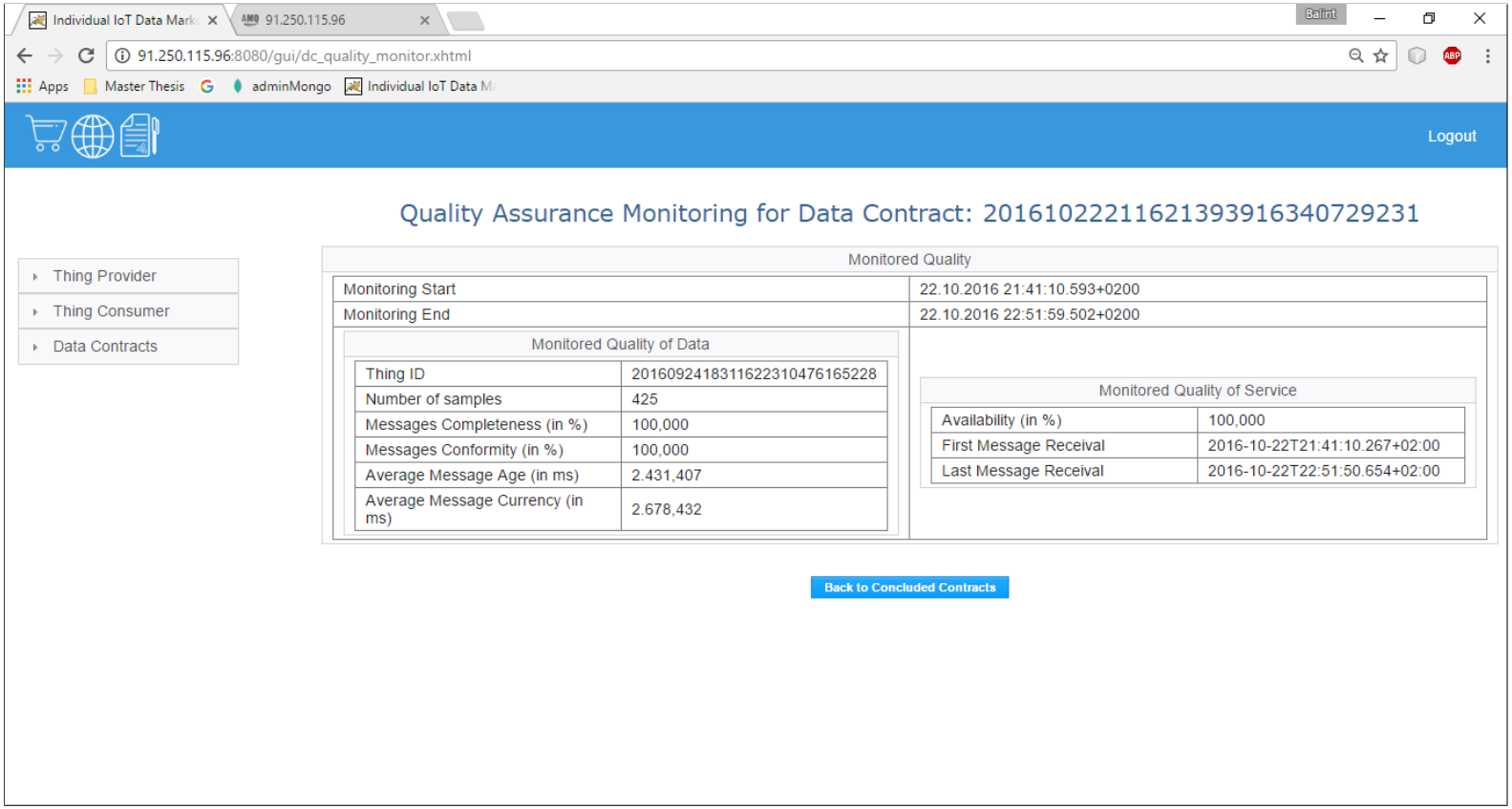


Figure 5.4: Monitoring Screenshot 1

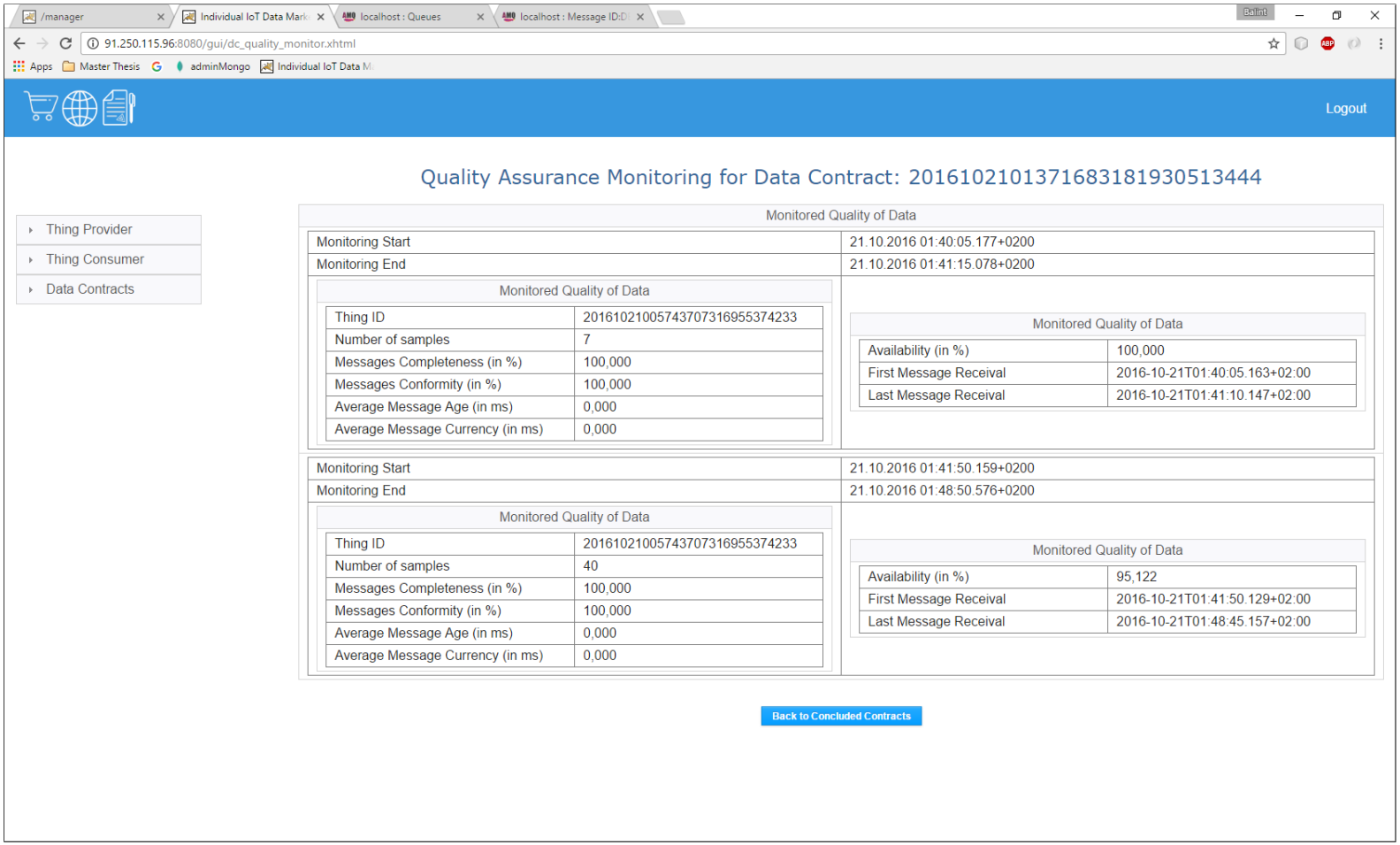


Figure 5.5: Monitoring Screenshot 2

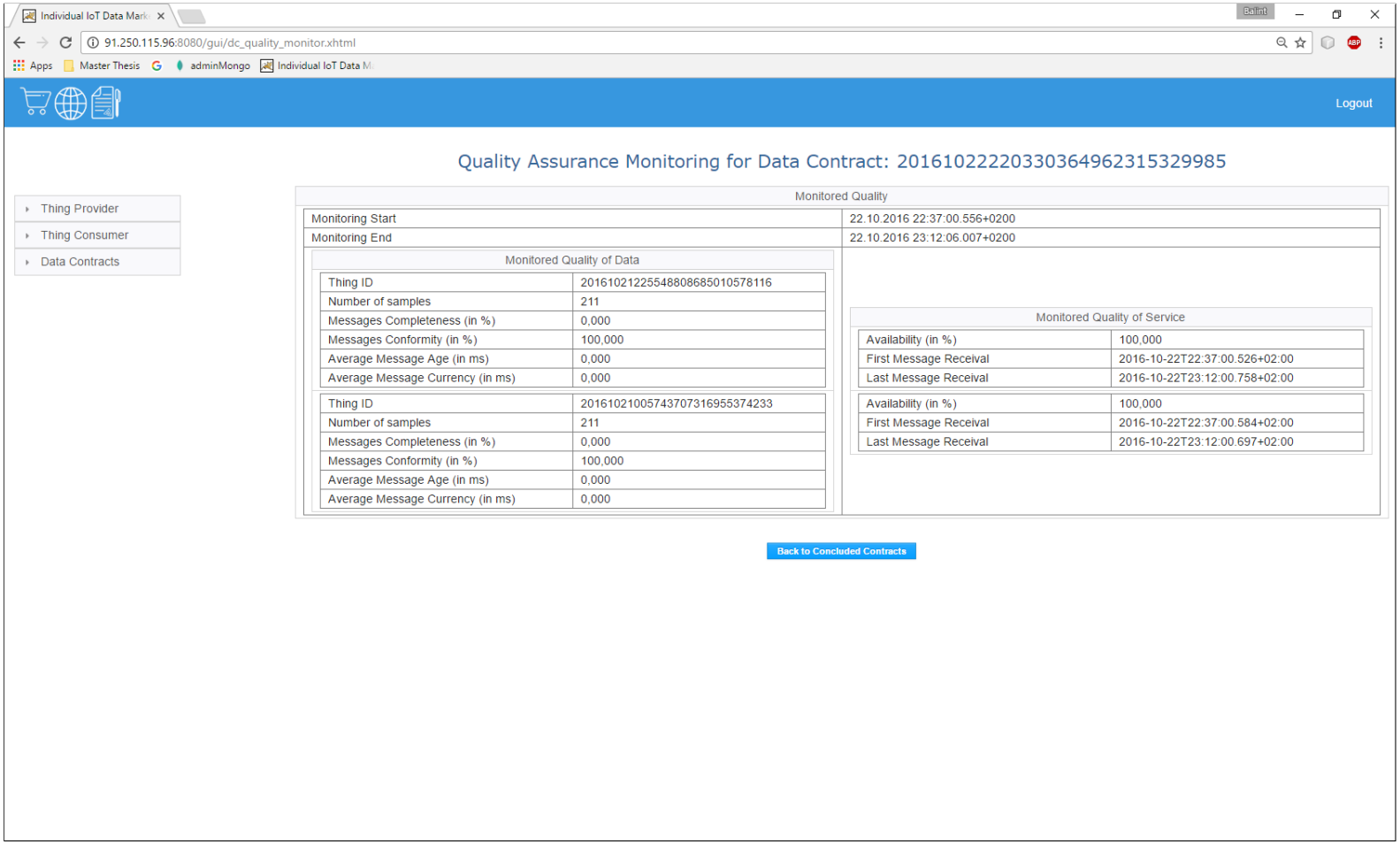


Figure 5.6: Monitoring Screenshot 3

5.2.3 Analytical Evaluation

In the previous section we analysed the prototype from an end user perspective and demonstrated its utility and the fact that it can serve as a possible solution from the problem described in Chapter 1. In this section we will conduct experiments in order to analyse the prototypes performance and stability, especially when running in limited environments such as those mentioned in Chapter 5.2.1.

First we conducted performance tests in environment 1 using the tool Apache JMeter [84]. These performance tests are based on the described representative scenarios from Chapter 3.2. All tests consisted of different REST requests, which are sending different JSON objects directly to the microservices and/or the *ServiceHandler* in order to simulate different requests from different GUI servers. For most of the conducted tests, the following properties were recorded automatically by the the tool Apache JMeter:

- Threads: The number of threads which was set manually for every performance test. One thread can be seen similar to one user.
- Ramp-up period: The time in seconds, during which all threads were started. E.g., if the number of threads is 10 and the ramp-up period is 10, then 10 threads were started during a period of 10 seconds.
- Loop count: The number of times the threads are restarted .
- Number of samples: The total number of requests which were sent (e.g., if a thread group consists of two requests and one thread is started to execute the thread group and the loop count is two, then the total number of sent samples is four).
- Average response time: The average time in milliseconds that was needed in order to receive a response for a request.
- Minimum response time: The minimum time in milliseconds that was needed in order to receive a response for a request.
- Maximum response time: The maximum time in milliseconds that was needed in order to receive a response for a request.
- Standard deviation: The standard deviation from the average response time.
- Error: How many requests (in percent) were responded with an error (e.g., 500 internal server error).
- Throughput: The number of processed requests (usually measured per second or minute).

User Registration Experiments

These performance tests were aimed at the *ServiceHandler* to evaluate the person registration REST service and were conducted in environment 1. When a person is registered, the GUI server sends a JSON object to a REST service. This JSON object contains all fields which were filled out in the GUI (e.g., for natural persons their first name, last name, e-mail, address, etc.). Similar JSON objects, like those created by the GUI server were constructed and used in the performance tests. However, in order to simulate different persons, some fields contain a reference to a JMeter variable (i.e., $\${RAND\_INT}$), which generates random integers.

A thread group was created, which consisted of two requests simulating the consecutive registration of two persons: one natural and one legal person. For the avoidance of redundancies in the database, a search process was implemented in the registration REST services. Consequently, before starting the tests, 10000 persons were added to the database in order to simulate a real running system.

During the first test, 10 threads were started during a period of 10 seconds, (i.e., ramp-up period) simulating the registration of the two persons 10 times which resulted in 200 person registrations. The average response time for the complete registration of a person was 56 milliseconds, the minimum response time was 43 milliseconds and the maximum 113 milliseconds. The standard deviation was 11,97 and all requests have been successfully processed without any errors. The recorded throughput (i.e., processed requests per second) was 19,8.

During the next performance tests, the number of threads was increased and the system behaviour was observed. All requests from all tests have been successfully processed without any errors indicating a stable system behaviour, even when the CPU load reached the maximum of 100%. The maximum CPU load was observed only for a few seconds. The system performance increased as the number of threads increased, however in the end it started decreasing as can be seen from the measured standard deviation values. All measured and observed values indicate a stable system behaviour even when high amounts of requests occur at the same time. The test results have been logged in Table 5.5.

Thing Registration Experiments

With the second set of performance tests (see Table 5.6), the *Thing* registration REST service was evaluated in environment 1. In this case single a thread group, consisting of a single request was created. This request accessed the *Thing* registration REST service directly.

All requests have been successfully processed and no errors occurred. At the beginning 10 threads were started, which were processed very fast and resulted in a low CPU increase. As the number of threads increased, the throughput also increased, the average response

Test Nr.	**1**	**2**	**3**	**4**	**5**
Threads	10	20	40	80	100
Ramp-Up Period (s)	10	10	10	10	10
Loop Count	10	10	10	10	10
Nr. of Samples	200	400	800	1600	2000
Average (ms)	56	60	73	128	258
Min (ms)	43	42	45	45	46
Max (ms)	113	121	218	623	3344
Standard Deviation	11,97	13,10	21,71	65,33	176,27
Error (%)	0,00	0,00	0,00	0,00	0,00
Throughput (s)	19,8	37,3	70,8	137,4	126,8
Max. Load CPU Core 1 (%)	1,3	2,0	46,0	84,0	98,3
Max. Load CPU Core 2 (%)	42,9	84,0	100,0	100,0	100,0
Load Memory (GB)	3,04	3,08	3,08	3,08	3,08

Table 5.5: Performance Test Results - Person Registration

time remained constant, the standard deviation fluctuated a little and the maximum CPU load increased for very short periods of time.

Test Nr.	**1**	**2**	**3**	**4**	**5**
Threads	10	20	40	80	100
Ramp-Up Period (s)	10	10	10	10	10
Loop Count	10	10	10	10	10
Nr. of Samples	100	200	400	800	1000
Average (ms)	53	53	50	63	53
Min (ms)	38	35	37	36	34
Max (ms)	126	127	113	361	353
Standard Deviation	17,31	18,01	13,94	30,23	21,91
Error (%)	0,00	0,00	0,00	0,00	0,00
Throughput (s)	10,04	19,5	39,0	77,0	94,6
Max. Load CPU Core 1 (%)	0,7	0,7	1,3	3,2	2,6
Max. Load CPU Core 2 (%)	29,0	37,0	60,3	64,9	90,9
Load Memory (GB)	3,05	3,05	3,05	3,06	3,08

Table 5.6: Performance Test Results - *Thing* Registration

Data Contract Negotiation Experiments

The third set of performance tests was aimed at the data contract negotiation REST service and was conducted in environment 1. In this case a thread group was created, which consisted of five requests, simulating two data contract offers, two counter offers and one acceptance. The first test was conducted using 10 threads and indicated a stable

system behaviour and fast response times (on average 59 milliseconds for a request). The test was repeated with 20, 40 and 80 threads and each time all requests were successfully processed. At the beginning a decrease of the standard deviation was observed and afterwards an increase, together with the minimum and maximum response time. The maximum CPU load for both set of performance tests was observed for very short periods of time.

The test results for both sets of performance tests (logged in Tables 5.6 and 5.7) have shown a stable system behaviour even when a higher amount of requests occurs.

Test Nr.	**1**	**2**	**3**	**4**
Threads	10	20	40	80
Ramp-Up Period (s)	10	10	10	10
Loop Count	10	10	10	10
Nr. of Samples	500	1000	2000	4000
Average (ms)	59	52	62	106
Min (ms)	36	36	37	43
Max (ms)	223	122	393	815
Standard Deviation	25,06	11,80	19,99	57,30
Error (%)	0,00	0,00	0,00	0,00
Throughput (s)	41,7	82,5	156,8	269,1
Max. Load CPU Core 1 (%)	1,3	1,3	23,8	88,9
Max. Load CPU Core 2 (%)	69,3	97,6	100,0	100,0
Load Memory (GB)	3,05	3,05	3,05	3,05

Table 5.7: Performance Test Results - Data Contract Negotiation

5.2.4 Data Contract Monitoring Experiments

At the beginning the resource consumption during the previously conducted descriptive analysis evaluation in environment 1 was observed. During the whole period of the conducted experiment the average CPU load remained below 3% for both cores. The maximum CPU load was 10% for core 1 and 100% for core 2. However, this load was observed for short periods of time lasting approximately 1 second. All observed values indicate a stable running system behaviour.

The second set of performance tests were conducted to evaluate the average computation time for the QoD and QoS for 100 data contracts, each consisting of one *Thing*, for 100 respectively 1000 messages produced by the three types of *Things* mentioned in the previous section. These performance tests run in environment 2 and were executed by calling the computation component directly (i.e., no messages were read/sent from/to any message broker).

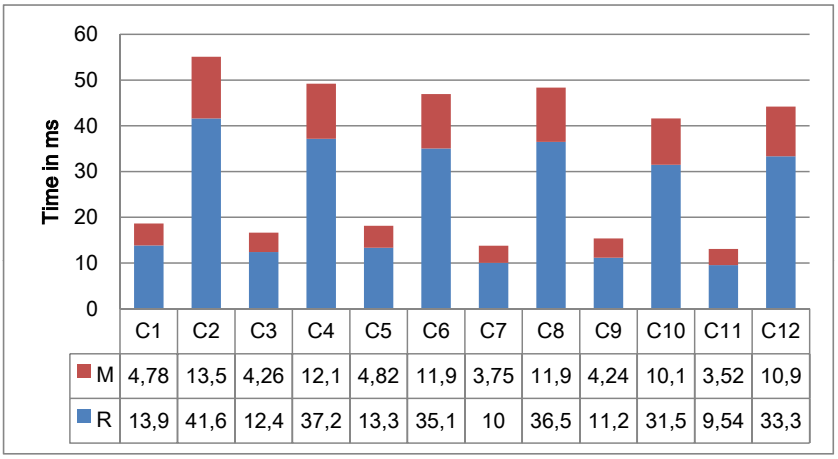
The experiments properties and results are as follows (where t is the *Thing* type: (1) temperature sensor, (2) device simulation with data from [82] and (3) device simulation

with data from [83]), n is the number of *Things*, m the number of produced messages per *Thing*, f the broadcasting frequency in ms and t_{avg} is the average computation time in ms of the QoD and QoS metrics for one single message):

- $t = 1, n = 100, f = 100ms, m = 100, t_{avg} = 1.46ms$
- $t = 1, n = 100, f = 100ms, m = 1000, t_{avg} = 1.07ms$
- $t = 2, n = 100, f = 100ms, m = 100, t_{avg} = 1.38ms$
- $t = 2, n = 100, f = 100ms, m = 1000, t_{avg} = 1.01ms$
- $t = 3, n = 100, f = 100ms, m = 100, t_{avg} = 1.77ms$
- $t = 3, n = 100, f = 100ms, m = 1000, t_{avg} = 1.20ms$

This results demonstrate that the computation of QoD and QoS metrics is not a time consuming task.

To further evaluate the performance of the monitoring component and the caused monitoring overhead in addition to simple message redistribution we conducted 12 further performance tests in environment 2 simulating real data transmissions and data contracts. For each of the previously mentioned *Thing* types 4 tests were conducted. In each of these 4 cases either 10 or 20 *Things* broadcasted data to either one or two message queues. In all cases each *Thing* was bound to a different data contract and for all *Things* a minimum number of 1000 messages were consumed. The results are shown in Figure 5.7 (where R stands for the time in milliseconds needed to read a message and redistribute it and M stands for the time used for the computation of the quality): the monitoring overhead ranges from approx. 32% to approx. 38%.



	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
M	4,78	13,5	4,26	12,1	4,82	11,9	3,75	11,9	4,24	10,1	3,52	10,9
R	13,9	41,6	12,4	37,2	13,3	35,1	10	36,5	11,2	31,5	9,54	33,3

Figure 5.7: Performance Test Results - Monitoring Overhead

The fact that all messages are first redistributed and afterwards, if enabled, the quality is computed, and the fact that all operations are in the milliseconds range (all below 60) proves that data stream monitoring is negligible when dealing with no real-time data transmissions.

5.2.5 Data Contract Recommendation Experiments

Before the execution of the data contract recommendation performance tests, 10000 persons and 10000 *Things* were inserted into the database. For each inserted *Thing* a random number (uniformly distributed integer value) from 1 to 1000 was generated. This generated number represented the number of ratings of a *Thing* (i.e., how many people submitted a rating for this *Thing*). For each of these ratings, a random user was picked and a random rating was generated and added to the *Thing*. This way, different *Things* with different ratings by different users were simulated. Each submitted recommendation request from each performance test was executed for one of the existent users (randomly picked from the existing 10000 users).

All performance tests were conducted in environment 2. In all cases, all requests were processed successfully (see Table 5.8 and Figure 5.8).

Test Nr.	**1**	**2**	**3**	**4**
Threads	10	20	40	80
Ramp-Up Period (s)	10	10	10	10
Loop Count	10	10	10	10
Nr. of Samples	100	200	400	800
Average (ms)	91	102	661	1431
Min (ms)	79	80	80	86
Max (ms)	163	169	4124	18874
Standard Deviation	14	17	492	3104
Error (%)	0,00	0,00	0,00	0,00
Throughput (min)	10,01	19,20	24,20	25,80

Table 5.8: Performance Test Results - *Thing* Recommendation

During each performance test the average CPU load was observed and remained below 20%. Furthermore in all cases a high discrepancy was observed between the minimum and the maximum response time of the requests. As the number of threads increased, the throughput remained constant (around 1 request per second), but the average response time increased, together with the standard deviation.

One possible explanation for the observations is because of the computation of the neighbourhood: in some cases a user has multiple neighbours and only those which are closest to him/her are taken into consideration. In order to do so, the distance to all neighbours has to be calculated (for more details see Chapter 2). One possible way to deal

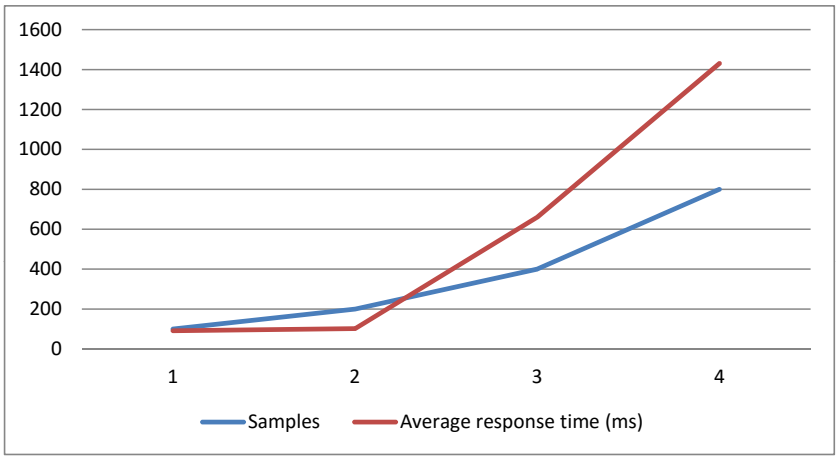


Figure 5.8: Performance Test Results - *Thing* Recommendation

with this high performance consuming task would be to pre-calculate the neighbourhood of a user, or perhaps even the whole recommendation and to save it into the database.

The results of the conducted experiments from this section (i.e., analytical evaluation) demonstrate the stability and performance of the implemented prototype, even when running in limited environments. Furthermore, they demonstrate that a large number of concurrent requests are supported and can be processed error free.

CHAPTER 6

Summary

In this thesis we introduced *IDAC* – a new extensible contract-aware IoT data marketplace framework, which was build as a scalable distributed system and which supports bilateral negotiations and establishment of data contracts by taking into account, amongst others, the following possible data contractual clauses: data rights (e.g., derivation, collection, reproduction, commercial usage), quality of data (e.g., completeness, conformity, etc.), quality of service (e.g., availability), pricing models, purchasing policy (e.g., contract termination) and control and relationship (e.g., warranty, jurisdiction).

One possible solution to monitor the data flow, is by measuring the quality of data and the quality of service. QoD measurements can be obtained by validating a message against the meta-model of a *Thing* and QoS measurements can be computed by taking into account the expected broadcasting frequency of a device. The evaluation results show that the monitoring of the data flow on an individual data contract level is not a time consuming task, even when running within limited environments.

Furthermore we have shown that custom components containing hybrid recommendation techniques can be plugged in and used to recommend the conclusion of data contracts for data produced by *Things*.

The obtained results demonstrate the effectiveness of the introduced framework for negotiating, monitoring and recommending data contracts in IoT dataspaces.

A future research direction in this area is the development of custom components and their integration with IoT data markets and the automated enforcement of data contractual clauses – e.g., the automated configuration of *Things* in accordance to data contracts (for example to automatically reconfigure the broadcasting frequency of a device, or to automatically modify the data structure of the produced data).

Bibliography

[1] G. E. Kersten, *Negotiations and E-negotiations: People, Models, and Systems*, 2010.

[2] G. E. Kersten and H. Lai, "Negotiation support and e-negotiation systems: An overview," *Group Decision and Negotiation*, vol. 16, no. 6, pp. 553–586, oct 2007. [Online]. Available: http://dx.doi.org/10.1007/s10726-007-9095-5

[3] P. Braun, J. Brzostowski, G. Kersten, J. B. Kim, R. Kowalczyk, S. Strecker, and R. Vahidov, "e-negotiation systems and software agents: Methods, models, and applications," in *Intelligent Decision-making Support Systems*. Springer London, 2006, pp. 271–300. [Online]. Available: http://dx.doi.org/10.1007/1-84628-231-4_15

[4] I. T. Union. "Internet of things global standards initiative." 2016. [Online]. Available: http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx. [Accessed: Feb. 10, 2016].

[5] Gartner. "Gartner says 6.4 billion connected "things" will be in use in 2016." Nov. 2015. [Online]. Available: http://www.gartner.com/newsroom/id/3165317. [Accessed: Feb. 10, 2016].

[6] Cisco. "Connections Counter: The Internet of Everything in Motion." Jul. 2016. [Online]. Available: http://newsroom.cisco.com/feature-content?articleId=1208342. [Accessed: Mar. 21, 2016].

[7] V. Turner, D. Reinsel, J. F. Gantz, and S. Minton. "The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things." 2014. [Online]. Available: http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm. [Accessed: Mar. 24, 2016].

[8] Tilepay. "tilepay." 2016. [Online]. Available: http://www.tilepay.org. [Accessed: Feb. 16, 2016].

[9] T.-D. Cao, T.-V. Pham, Q.-H. Vu, H.-L. Truong, D.-H. Le, and S. Dustdar, "MARSA: A Marketplace for Realtime Human-Sensing Data," 2016, transactions on Internet Technology, 2016. Accepted.

[10] BDEX. "Bdex, the marketplace for data." 2016. [Online]. Available: http://www.bigdataexchange.com. [Accessed: Mar. 21, 2016].

[11] Microsoft. “Microsoft Azure Marketplace.” 2016. [Online]. Available: https://datamarket.azure.com/browse/data. [Accessed: Mar. 22, 2016].

[12] M. Franklin, A. Halevy, and D. Maier, “From databases to dataspaces: A new abstraction for information management,” *SIGMOD Rec.*, vol. 34, no. 4, pp. 27–33, Dec. 2005. [Online]. Available: http://doi.acm.org/10.1145/1107499.1107502

[13] F. B. Balint and H.-L. Truong, “On Supporting Contract-aware IoT Dataspace Services,” 2016, submitted.

[14] The Member States of the European Union. “Consolidated version of the Treaty on European Union.” 2012. [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A12012M%2FTXT. [Accessed: Aug. 15, 2016].

[15] ——. “Charter of Fundamental Rights of the European Union.” 2012. [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:12012P/TXT. [Accessed: Aug. 16, 2016].

[16] European Parliament, Council of the European Union. “Directive (EU) 2016/680 of the European Parliament and of the Council.” 2016. [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016L0680. [Accessed: Aug. 9, 2016].

[17] European Commission. “Protection of personal data.” 2016. [Online]. Available: http://ec.europa.eu/justice/data-protection/. [Accessed: Aug. 11, 2016].

[18] European Parliament, Council of the European Union. “Directive 2000/31/EC of the European Parliament and of the Council.” 2016. [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32000L0031. [Accessed: Aug. 9, 2016].

[19] Council of the European Union. “Proposal for a directive of the European Parliament and of the Council on certain aspects concerning contracts for the supply of digital content.” 2015. Procedure 2015/0287/COD. [Online]. Available: http://data.consilium.europa.eu/doc/document/ST-15251-2015-INIT/en/pdf. [Accessed: Aug. 15, 2016].

[20] European Parliament, Council of the European Union. “Regulation (EC) No 593/2008 of the European Parliament and of the Council of 17 June 2008 on the law applicable to contractual obligations (Rome I).” 2008. [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A12012M%2FTXT. [Accessed: Aug. 16, 2016].

[21] ——. “Regulation (EU) No 1215/2012 of the European Parliament and of the Council of 12 December 2012 on jurisdiction and the recognition and enforcement of judgments in civil and commercial matters.” 2012. [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex%3A32012R1215. [Accessed: Aug. 16, 2016].

[22] "Bundesrecht konsolidiert: Gesamte Rechtsvorschrift für Datenschutzgesetz 2000." 2016. [Online]. Available: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10001597. [Accessed: Aug. 15, 2016].

[23] "Bundesrecht konsolidiert: Gesamte Rechtsvorschrift für E-Commerce-Gesetz." 2016. [Online]. Available: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=20001703. [Accessed: Aug. 15, 2016].

[24] "Urheberrechtsgesetz (UrhG)." 2016. [Online]. Available: http://www.jusline.at/Urheberrechtsgesetz_(UrhG).html. [Accessed: Aug. 15, 2016].

[25] "Allgemeines Bürgerliches Gesetzbuch (ABGB)." 2016. [Online]. Available: https://www.jusline.at/Allgemeines_Buergerliches_Gesetzbuch_(ABGB).html. [Accessed: Aug. 15, 2016].

[26] M. Schoop, J. Köller, T. List, and C. Quix, "A three-phase model of electronic marketplaces for software components in chemical engineering," in *Towards the E-Society*. Springer US, 2001, pp. 506–522. [Online]. Available: http://dx.doi.org/10.1007/0-306-47009-8_37

[27] L. Thompson, *The Mind and Heart of the Negotiator*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2000.

[28] D. K. W. Chiu, S. C. Cheung, P. C. K. Hung, S. Y. Y. Chiu, and A. K. K. Chung, "Developing e-negotiation support with a meta-modeling approach in a web services environment," *Decis. Support Syst.*, vol. 40, no. 1, pp. 51–69, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.dss.2004.04.004

[29] H. Raiffa, *The Art and Science of Negotiation*. Belknap Press, 1985. [Online]. Available: http://www.amazon.com/Art-Science-Negotiation-Howard-Raiffa/dp/067404813X%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D067404813X

[30] G. E. Kersten, S. E. Strecker, and K. P. Law, "Protocols for electronic negotiation systems: Theoretical foundations and design issues," in *E-Commerce and Web Technologies*. Springer Berlin Heidelberg, 2004, pp. 106–115. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30077-9_11

[31] S. Strecker, G. Kersten, J.-B. Kim, and K. P. Law, "Electronic negotiation systems: The invite prototype," in *Multikonferenz Wirtschaftsinformatik 2006*, Gesellschaft für Informatik e.V. Berlin: GITO, 20.-22. Februar 2006, pp. 315–331.

[32] M. Schoop, A. Jertila, and T. List, "Negoisst: a negotiation support system for electronic business-to-business negotiations in e-commerce," *Data & Knowledge Engineering*, vol. 47, no. 3, pp. 371–401, dec 2003. [Online]. Available: http://dx.doi.org/10.1016/S0169-023X(03)00065-X

[33] O. Marjanovic and Z. Milosevic, "Towards formal modeling of e-contracts," in *Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International*, 2001, pp. 59–68.

[34] R. Y. K. Lau, "Towards a web services and intelligent agents-based negotiation system for b2b ecommerce," *Electron. Commer. Rec. Appl.*, vol. 6, no. 3, pp. 260–273, Oct. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.elerap.2006.06.007

[35] J. B. Kim and A. Segev, "A web services-enabled marketplace architecture for negotiation process management," *Decis. Support Syst.*, vol. 40, no. 1, pp. 71–87, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.dss.2004.04.005

[36] M. Benyoucef and R. K. Keller, "A conceptual architecture for a combined negotiation support system," in *Database and Expert Systems Applications, 2000. Proceedings. 11th International Workshop on*, 2000, pp. 1015–1019.

[37] M. Benyoucef and S. Rinderle, "Modeling e-negotiation processes for a service oriented architecture," *Group Decision and Negotiation*, vol. 15, no. 5, pp. 449–467, aug 2006. [Online]. Available: http://dx.doi.org/10.1007/s10726-006-9038-6

[38] P. R. Krishna and K. Karlapalem, "Electronic contracts," *IEEE Internet Computing*, vol. 12, no. 4, pp. 60–68, July 2008.

[39] H.-L. Truong, M. Comerio, F. D. Paoli, G. R. Gangadharan, and S. Dustdar, "Data Contracts for Cloud-based Data Marketplaces," *Int. J. Comput. Sci. Eng.*, vol. 7, no. 4, pp. 280–295, Oct. 2012. [Online]. Available: http://dx.doi.org/10.1504/IJCSE.2012.049749

[40] H.-L. Truong, G. Gangadharan, M. Comerio, S. Dustdar, and F. De Paoli, "On Analyzing and Developing Data Contracts in Cloud-Based Data Marketplaces," in *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, Dec 2011, pp. 174–181.

[41] P. Kumari, A. Pretschner, J. Peschla, and J.-M. Kuhn, "Distributed data usage control for web applications: A social network implementation," in *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '11. New York, NY, USA: ACM, 2011, pp. 85–96. [Online]. Available: http://doi.acm.org/10.1145/1943513.1943526

[42] U. Congress. "Sarbanes-oxley act of 2002." [Online]. Available: https://www.gpo.gov/fdsys/pkg/PLAW-107publ204/html/PLAW-107publ204.htm. [Accessed: Nov. 1, 2016].

[43] S. Bradshaw, C. Millard, and I. Walden, "Contracts for clouds: comparison and analysis of the terms and conditions of cloud computing services," *International Journal of Law and Information Technology*, vol. 19, no. 3, pp. 187–223, 2011.

[44] S. Sen, C. Joe-Wong, S. Ha, and M. Chiang, "A survey of smart data pricing: Past proposals, current plans, and future trends," *ACM Comput. Surv.*, vol. 46, no. 2, pp. 15:1–15:37, Nov. 2013. [Online]. Available: http://doi.acm.org/10.1145/2543581.2543582

[45] A. Goodchild, C. Herring, and Z. Milosevic, "Business contracts for b2b." *ISDO*, vol. 30, 2000.

[46] C. Batini and M. Scannapieco, *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Springer, 2006.

[47] W. W. Eckerson. "Data quality and the bottom line." [Online]. Available: http://download.101com.com/pub/tdwi/Files/DQReport.pdf. [Accessed: Jul. 22, 2016].

[48] R. Y. Wang and D. M. Strong, "Beyond accuracy: What data quality means to data consumers," *Journal of Management Information Systems*, vol. 12, no. 4, pp. 5–33, 1996. [Online]. Available: http://dx.doi.org/10.1080/07421222.1996.11518099

[49] L. L. Pipino, Y. W. Lee, and R. Y. Wang, "Data quality assessment," *Commun. ACM*, vol. 45, no. 4, pp. 211–218, Apr. 2002. [Online]. Available: http://doi.acm.org/10.1145/505248.506010

[50] C. Fürber and M. Hepp, "Towards a vocabulary for data quality management in semantic web architectures," in *Proceedings of the 1st International Workshop on Linked Web Data Management*, ser. LWDM '11. New York, NY, USA: ACM, 2011, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/1966901.1966903

[51] K.-U. Sattler, *Data Quality Dimensions*. Boston, MA: Springer US, 2009, pp. 612–615. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-39940-9_108

[52] S. Systems. "Data quality glossary – conformity." May 2011. [Online]. Available: http://www.dqglossary.com/conformity.html. [Accessed: Jul. 24, 2016].

[53] C. Tărâță. "Data quality dimensions – from accuracy to uniqueness." Mar. 2015. [Online]. Available: http://www.performancemagazine.org/data-quality-dimensions-from-accuracy-to-uniqueness/. [Accessed: Jul. 24, 2016].

[54] V. S. Council. "Spatial information data quality guidelines." Sep. 2009. [Online]. Available: http://victorianspatialcouncil.org/cms/library/attachments/SIMF%20Data%20Quality%20Guidelines%20Edition%202_September%202009.pdf. [Accessed: Jul. 24, 2016].

[55] O. Adinolfi, R. Cristaldi, L. Coppolino, and L. Romano, "Qos-monaas: A portable architecture for qos monitoring in the cloud," in *Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on*, Nov 2012, pp. 527–532.

[56] R. Duan, X. Chen, and T. Xing, "A QoS Architecture for IOT," in *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, Oct 2011, pp. 717–720.

[57] D. A. Menasce, "QoS issues in Web services," *IEEE Internet Computing*, vol. 6, no. 6, pp. 72–75, Nov 2002.

[58] Microsoft. "What Is QoS." 2003. [Online]. Available: https://technet.microsoft.com/en-us/library/cc757120(v=ws.10).aspx. [Accessed: Jul. 21, 2016].

[59] Cisco. "QoS Frequently Asked Questions." 2009. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-policing/22833-qos-faq.html. [Accessed: Jul. 21, 2016].

[60] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds., *Recommender Systems Handbook*. Springer, 2010. [Online]. Available: http://www.amazon.com/Recommender-Systems-Handbook-Francesco-Ricci/dp/0387858199%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0387858199

[61] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in *The adaptive web*. Springer, 2007, pp. 291–324.

[62] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 43–52. [Online]. Available: http://dl.acm.org/citation.cfm?id=2074094.2074100

[63] C. Desrosiers and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods," in *Recommender systems handbook*. Springer, 2011, pp. 107–144.

[64] X. Amatriain, A. Jaimes, N. Oliver, and J. M. Pujol, "Data mining methods for recommender systems," in *Recommender Systems Handbook*. Springer, 2011, pp. 39–71.

[65] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," in *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, ser. CSCW '94. New York, NY, USA: ACM, 1994, pp. 175–186. [Online]. Available: http://doi.acm.org/10.1145/192844.192905

[66] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An Algorithmic Framework for Performing Collaborative Filtering," in *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in*

Information Retrieval, ser. SIGIR '99. New York, NY, USA: ACM, 1999, pp. 230–237. [Online]. Available: http://doi.acm.org/10.1145/312624.312682

[67] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based Collaborative Filtering Recommendation Algorithms," in *Proceedings of the 10th International Conference on World Wide Web*, ser. WWW '01. New York, NY, USA: ACM, 2001, pp. 285–295. [Online]. Available: http://doi.acm.org/10.1145/371920.372071

[68] P. Lops, M. De Gemmis, and G. Semeraro, "Content-based recommender systems: State of the art and trends," in *Recommender systems handbook*. Springer, 2011, pp. 73–105.

[69] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The adaptive web*. Springer, 2007, pp. 325–341.

[70] K. Akpınar, K. A. Hua, and K. Li, "ThingStore: A Platform for Internet-of-things Application Development and Deployment," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '15. New York, NY, USA: Association for Computing Machinery (ACM), 2015, pp. 162–173. [Online]. Available: http://dx.doi.org/10.1145/2675743.2771833

[71] Amazon Web Services. "AWS Public Data Sets." 2016. [Online]. Available: https://aws.amazon.com/datasets/. [Accessed: Mar. 22, 2016].

[72] MathWorks, Inc. "Thingspeak, The open data platform for the Internet of Things." 2016. [Online]. Available: https://thingspeak.com/. [Accessed: Mar. 21, 2016].

[73] Array of Things. "Array of Things." 2016. [Online]. Available: https://arrayofthings.github.io/. [Accessed: Sep. 28, 2016].

[74] O. Boris, J. Jan, S. Jochen, A. Sören, M. Nadja, W. Sven, and C. Jan. "Industrial Data Space." Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. 2016. [Online]. Available: www.fraunhofer.de/content/dam/zv/de/Forschungsfelder/industrial-data-space/Industrial-Data-Space_whitepaper.pdf. [Accessed: Sep. 28, 2016].

[75] D. Le, N. C. Narendra, and H. L. Truong, *HINC - Harmonizing Diverse Resource Information across IoT, Network Functions, and Clouds*, 2016.

[76] MongoDB Inc. "Introduction to MongoDB." 2016. [Online]. Available: https://docs.mongodb.com/manual/introduction/. [Accessed: Sep. 28, 2016].

[77] Apache Software Foundation. "Apache ActiveMQ." 2016. [Online]. Available: http://activemq.apache.org/. [Accessed: Nov. 13, 2016].

[78] ——. "Apache Camel." 2016. [Online]. Available: https://camel.apache.org/. [Accessed: Nov. 13, 2016].

[79] PrimeTek Informatics. “PrimeFaces.” 2016. [Online]. Available: http://primefaces.org/. [Accessed: Nov. 13, 2016].

[80] Apache Software Foundation. “Apache Tomcat.” 2016. [Online]. Available: http://tomcat.apache.org/. [Accessed: Nov. 13, 2016].

[81] ——. “Apache Maven.” 2016. [Online]. Available: http://maven.apache.org/. [Accessed: Nov. 30, 2016].

[82] City of Austin open data. “Water Quality Sampling Data.” City of Austin. 2016. [Online]. Available: https://data.austintexas.gov/Environmental/Water-Quality-Sampling-Data/5tye-7ray. [Accessed: Oct. 14, 2016].

[83] OpenSignal. “OpenSignal.” OpenSignal. 2016. [Online]. Available: https://opensignal.com/. [Accessed: Oct. 14, 2016].

[84] Apache Software Foundation. “Apache JMeter.” 2016. [Online]. Available: https://jmeter.apache.org/. [Accessed: Nov. 13, 2016].