Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Context-Aware Data Validation for Machine Learning Pipelines

Tim Schubert

Course of Study:	Autonomous Systems
Examiner:	Prof. Dr. rer. nat. habil. Holger Schwarz
Supervisor:	Daniel Del Gaudio, M.Sc.
Commenced:	June 13, 2022
Completed:	December 13, 2022

Abstract

These days, machine learning plays a key role in plenty of applications. Self-learning algorithms are developed in not only industrial applications, e.g., production lines, or fleet management, but also in the private sector, e.g. smart homes. The performance of these programs is significantly related to the provided training data. A major challenge is preserving high quality of the data. Therefore, the demand for good data cleaning methods has been increasing over the past few years. While existing cleaning techniques can consider constraints and dependencies in data, they can not exploit context information automatically. Thus, they usually fail to track shifts in the data distributions or the associated error profiles. To overcome these limitations, this thesis introduces a novel pipeline for automated tabular data cleaning powered by dynamic functional dependency rules extracted from a context model. This context model is a live updating ontology, representing the current state of the environment where the data originates from. The proposed concept divides the pipeline into three main steps: (i) context modeling, (ii) dependency extraction, and (iii) data cleaning. As a proof-of-concept and for evaluation purposes, a prototype has been implemented. This prototype is evaluated on two different datasets, including an IoT dataset from a smart home use case and a commonly used benchmark dataset with different metrics from hospitals in the US. The evaluation shows that the proposed concept and pipeline for the data validation process performs better than typical state-of-the-art error detection methods.

Kurzfassung

Heutzutage spielt maschinelles Lernen in vielen Anwendungen eine Schlüsselrolle. Selbstlernende Algorithmen werden nicht nur für industrielle Anwendungen, z. B. in der Produktion oder im Flottenmanagement, sondern auch für den privaten Bereich, z. B. Smart Homes, entwickelt. Die Leistung dieser Programme hängt wesentlich von den bereitgestellten Trainingsdaten ab. Eine große Herausforderung besteht darin, Daten in hoher Qualität zu sammeln und diese zu erhalten. Daher ist die Nachfrage nach guten Fehlererkennungsmethoden in den letzten Jahren gestiegen. Bestehende Verfahren zu Fehlerbeseitigung können zwar Einschränkungen und Abhängigkeiten in den Daten berücksichtigen, aber sie können diese Kontextinformationen nicht automatisch nutzen. Daher sind sie in der Regel nicht in der Lage, Verschiebungen von Datenverteilungen oder die damit verbundenen Fehler zu erfassen. Um diese Einschränkungen zu überwinden, wird in dieser Arbeit eine neuartige Methode für die automatische Fehlererkennung und -verbesserung von Datensätzen vorgestellt. Diese basiert auf dynamischen Abhängigkeitsregeln, die aus einem Kontextmodell extrahiert werden. Das Kontextmodell ist eine sich ständig aktualisierende Ontologie, die den aktuellen Zustand der Umgebung widerspiegelt. Das eingeführte Konzept unterteilt die Datenvalidierung in drei Hauptschritte: (i) Kontextmodellierung, (ii) automatische Extraktion von Abhängigkeiten und (iii) Datenbereinigung. Als Proof-of-Concept und zu Evaluierungszwecken wurde ein Prototyp implementiert. Dieser Prototyp wurde an zwei verschiedenen Datensätzen evaluiert, darunter ein IoT Datensatz aus einem Smart-Home-Anwendungsfall und ein häufig verwendeter Benchmark-Datensatz mit verschiedenen Metriken über Krankenhäuser in den Vereinigten Staaten von Amerika. Die Evaluierung zeigt, dass das vorgeschlagene Konzept und die Pipeline für den Datenvalidierungsprozess besser abschneidet als dem Stand der Technik entsprechende Fehlererkennungsmethoden.

Contents

List of Figures

List of Tables

Listings

Acronyms

DDlog Differential Datalog. 49

FD Functional Dependency. 21, 24, 25, 26, 27, 28, 39

FN False Negative. 61, 62, 67, 68

FP False Positive. 61, 62, 67, 68

IDC International Data Corporation. 15, 22

IoT Internet of Things. 3, 4, 7, 9, 15, 16, 17, 18, 19, 21, 22, 23, 24, 26, 28, 29, 31, 32, 33, 34, 35, 36, 37, 38, 40, 41, 42, 43, 44, 51, 53, 54, 55, 59, 61, 65, 66, 67, 68, 69, 70, 71, 72

IP Internet Protocol. 34

MBP Multi-purpose Binding and Provisioning Platform. 29, 32, 33, 34, 47, 54

OFD Ontology Functional Dependency. 5, 7, 9, 21, 24, 25, 26, 28, 31, 32, 37, 38, 44, 45, 47, 48, 49, 50, 52, 56, 57, 58, 62, 63, 64, 65, 66, 69, 70, 71, 72

RDF Resource Description Framework. 44, 58

RFID Radio Frequency IDentification. 15

SPARQL SPARQL Protocol And RDF Query Language. 44, 45, 46, 47, 58, 59, 71

SSN Semantic Sensor Network. 9, 29, 33, 35, 36, 37

TCP Transmission Control Protocol. 43

TN True Negative. 62, 67, 68

TP True Positive. 61, 62, 67, 68

UNECE United Nations Economic Commission for Europe. 31

W3C World Wide Web Consortium. 35, 36, 44, 58

WSN Wireless Sensor Network. 15

1 Introduction

In today's digital world, the importance of data has heavily increased [BCC20]. Data has become a valuable and powerful asset in the Industrial era. Massive amounts of data can be researched to reveal hidden patterns or other secret correlations [SS13]. One of the core business tasks of advanced data usage is the support of business decisions [Bec16]. While Big Data influences the validity of data-driven decision-making, its usage has a big impact in these sectors. For example, applications for predictive analysis in maintenance are leading to new business models, as the manufacturers of machinery are in the best position to provide Big Data-based maintenance [Bec16]. With autonomous data-sensing technologies, like Wireless Sensor Networks (WSNs) or Radio Frequency IDentification (RFID), Big Data analysis has gained success and potential [SS13].
The Internet of Things (IoT) is a collective term for those devices, which are highly interconnected and produce, interchange and consume data. According to the International Data Corporation (IDC), there will be even more data generated in the future[1]. The data volume produced by IoT devices in 2015 will increase by more than five times in 2025 to 79 billion petabytes worldwide. This number likely correlates with the increasing number of IoT devices itself (Figure 1.1). By 2025, approximately 75 billion devices will be interconnected worldwide [MF16]. According to Forbes, this trend will continue to grow exponentially[2].

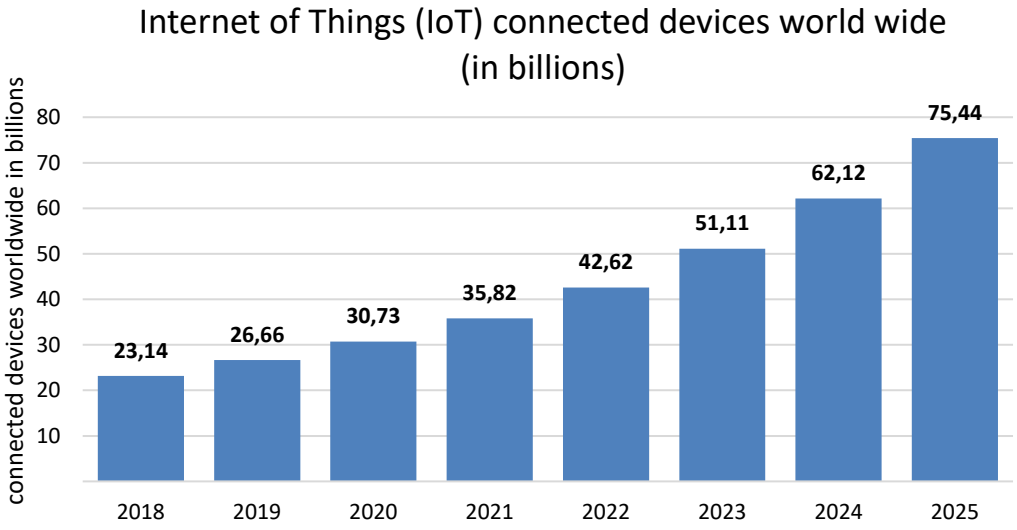

Figure 1.1: Connected devices worldwide from 2018 till 2025 [MF16]

[1] https://www.statista.com/statistics/1017863/worldwide-iot-connected-devices-data-size/
[2] https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-marketestimates-2016/#6a558beb292d

Big Data can be characterized in three core components: (a) *variety*, (b) *velocity* and (c) *volume* [SS13]. While the increasing size of data now outstrips traditional store and analysis techniques, velocity is a key element and very important for time limited processes. Big Data becomes even more challenging when adding variety [SS13]. With the huge amounts of data from various sources, the probability that some of those sources contain dirty data is very high [RD00]. Especially when combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct [3]. Other impacts include customer dissatisfaction, increased operational cost, less effective decision-making, and a reduced ability to make and execute a strategy [Red98]. In the long-term, poor data quality breeds organizational mistrust and makes it more difficult to align the enterprise [Red98]. The quality of data becomes one of the differentiating factors among businesses and the first line of defense in producing value from raw input data. Ensuring the quality of the data with respect to business and integrity constraints has become more important than ever [DEE+13]. Over the past few years, this has resulted in a surge of interest from both industry and academia on data cleaning problems. New abstractions, interfaces, approaches for scalability, and statistical techniques have been developed [CIKW16].

Due to the interconnection of IoT devices in large networks, the probability of a transmission or data error is increased [SDSB19]. Especially, the use of low power microchips and wireless communication leads to an increased number of errors or device failures in IoT environments. A faulty sensor can have a big negative impact if the data is further used, e.g., in machine learning pipelines.

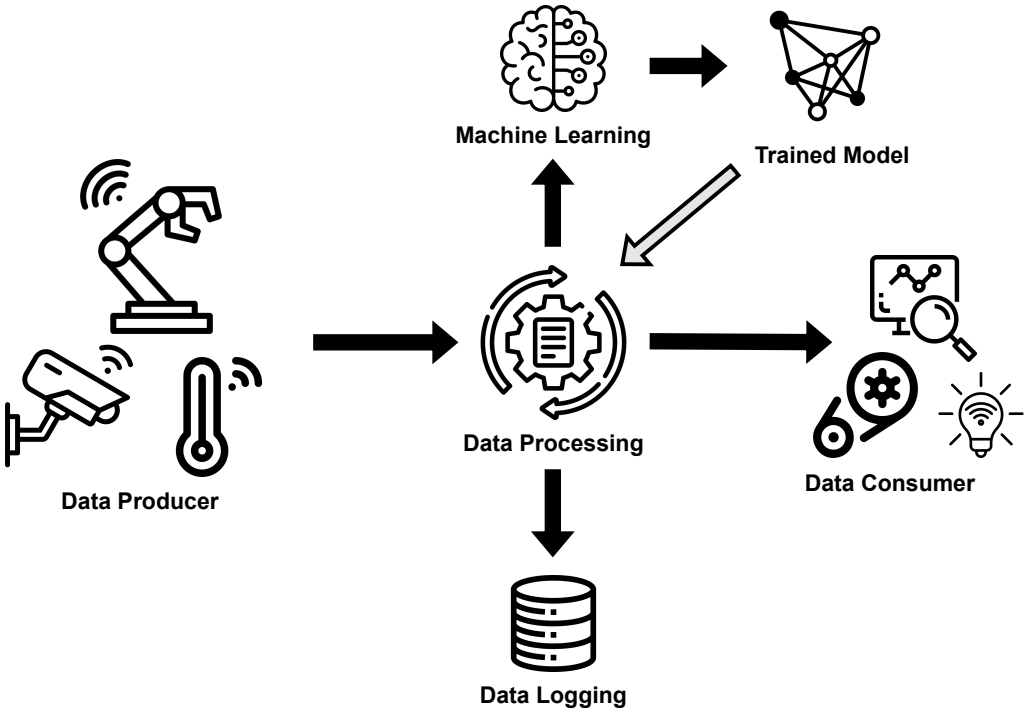

Figure 1.2: Example of IoT pipeline with machine learning

[3] https://www.tableau.com/learn/articles/what-is-data-cleaning

Figure 1.2 depicts a general structure how such environment could look like. Data producing devices, e.g., sensors, are builds the source of data flow pipelines. This data can then be processed to extract a model with machine learning algorithms or do some other action, e.g., trigger other actuators. False sensor values lead to faulty machine learning models, which will then impact the capabilities of this model in e.g. prediction. Furthermore, such errors can propagate through the whole data pipeline, which leads to wrong actions in a fully automated process. The task of error detection is very cumbersome for humans and thus not suitable for real-time applications or large datasets. Error detection algorithms can automate this process. While the state-of-the-art in these tools is quite advanced, they do typically still require substantial manual effort or programming [RD00]. Algorithms might mislabel correct values as false and vice versa because the overall understanding of the environment and context is missing. Metadata, reflected in schemata, is often insufficient to assess the data quality of a source [RD00]. Ontologies can provide a wide range of expert knowledge about the context while still being easy to understand and extensible if needed. The IoT Context Model, introduced by Del Gaudio et. al. is an example for such collection of knowledge about an IoT environment [DABS22]. Information about the spatial distribution of similar sensors can help in outlier detection, and technical details of a sensor can validate the plausibility of recorded values. A concept to automate the extraction of useful information from ontologies can improve error detection algorithms with less or none human involvement, which is the goal of this thesis.

1.1 Motivation Scenario

The following example will help to motivate the necessity of a good error detection method in machine learning pipelines.

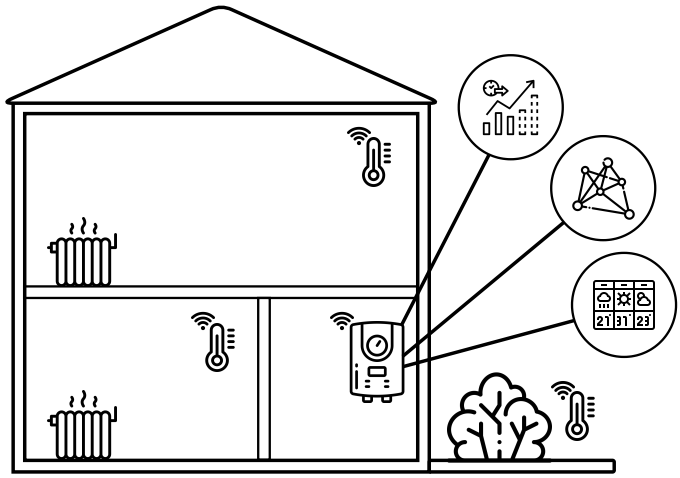

Figure 1.3: Smart Home with smart heater and prediction on energy costs

Figure 1.3 depicts a smart home application with three wireless temperature sensors. Two of them are mounted inside in different rooms and one sensor is placed outside. All temperature measurements are collected and stored to a database on the smart heating system in the house. Including current weather forecasts and the historical data about heating power and temperature

measurements, a machine learning model should be learned. This model then should predict the energy consumption for heating in the house for the next few days. Since all connections are wireless, transmission errors are one of many possible reasons for faulty data in the database. When learning a model, algorithms will always assume correct training data. The machine learning process will not differentiate if the value is correct or false, when building the model. Prediction with this model, based on wrong historical data, is likely to be wrong as well. To improve the prediction, one need to make sure that the dataset is error-free. Thus, a good error detection method is needed before the faulty data is inserted in to the machine learning pipeline.
In the scenario described above, a general application of the same error detection method for every temperature sensor, e.g., outlier detection, is not optimal. The maximum possible change rate of the temperature is drastically different, if taken inside or outside a house. Knowledge about the environment, like the sensor placement, can eventually help to improve error detection even further.

1.2 Goals of this Thesis

The main goal of this thesis is to improve existing error detection algorithms with the use of knowledge about the data context. In order to reach this goal, sub-goals are defined as followed:

- Introduction to error detection methods and how they are limited if no context is used
- Explanation how expert knowledge about a specific domain or environment can be described in ontologies, e.g., the IoT Context Model [DABS22]
- Extend the IoT context model with information about the context that is useful for error detection
- Development of a concept to extract dependencies from ontologies and to use them in the error detection process
- Implementation of a prototype to show how such concept can be used with current state-of-the-art frameworks for error detection
- Evaluation of the proposed concept using the implementation and real-world datasets

Structure

This thesis is structured as follows:

Chapter 2 - Fundamentals

Fundamentals which are necessary for this thesis are described in this chapter. A introduction to the IoT and ontologies is given.

Chapter 3 - Related Work

Chapter 3 gives an overview of related scientific work.

Chapter 4 - Context-Aware Data Validation

This chapter is the main part of this thesis and described the developed concept to include context-awareness in the data validation process. The context is gathered from ontologies which represent the domain expert knowledge.

Chapter 5 - Implementation

As a proof-of-concept, a prototype has been implemented. Chapter 5 describes the implementation in Python[4] using the error-detection framework HoloClean [RCIR17].

Chapter 6 - Evaluation

In this chapter, an evaluation of the prototypical implementation in Chapter 5 and the overall concept in Chapter 4 is given.

Chapter 7 - Summary and Future Work

Chapter 7 concludes this thesis and gives a summary of potential future work.

[4] https://www.python.org/about/

2 Fundamentals

The following chapter describes the fundamentals and basic terms, necessary for the comprehension of the presented concepts in Chapter 4. Firstly, an introduction to the IoT and smart devices is given (Section 2.1). Section 2.2 describes Big Data in general, and the challenge that arises when dealing with such big amounts of data. Typical errors in IoT datasets are sub-categorized and introduced in Section 2.3. The definition of ontologies (Section 2.4), particularly, the IoT Context Model introduced by Del Gaudio et. al. [DABS22] is given in Section 4.3.1. Further, the terms Functional Dependency (FD) (Section 2.5.1) and Ontology Functional Dependency (OFD) (Section 2.5.2) are explained, which form the basis for the concepts of dependency extraction from an ontology, the main contribution in this thesis (Section 4.4.3).

2.1 Internet of Things

The Internet of Things (IoT) is a collective term for technologies and describes the development and increasing networking of intelligent objects, so-called smart devices. A variety of heterogeneous devices are connected via standardized internet protocols to perform a wide range of tasks together, such as automating everyday activities. The focus is on the machine-to-machine communication, i.e. that devices communicate with each other without the need for human involvement [LN10]. Smart devices are often equipped with sensors or actors and are able to connect to the internet to offer services for a wide variety of users. Sensors record physical data from their environment, like temperature, humidity or their current location [DP11]. This data can be analyzed and processed by other devices, e.g., to control corresponding actuators. IoT can not only be used in private households, as described in Section 1.1, but also has many industrial applications.

The so-called Industry 4.0 is nowadays characterized by the use of IoT devices [Sin17]. The advantages of IoT devices, like the small size, wireless operation and low energy consumption, is useful for the application in existing production machinery. For example, through the acquisition of measurement data, a real-time evaluation of measurements can take place. This allows operators to precisely react to certain events or warnings, while still having an overview over all the relevant data. In general, the interconnection of devices speeds up productivity and responses to faults in production processes.
IoT applications in the private sector are mainly dominated by Smart Homes. Electronic devices in a household, like lights, doors or the heater control system can, when connected to each other, execute arbitrary automations. The connection to the internet, gives the user worldwide full control of the devices. Current development in compactness and power efficiency of microprocessors, as well as advances in wireless communication and battery technology, builds the foundation for a variety of use cases for IoT devices. Thus, they can be deployed almost everywhere. This flexibility is the reason why to expect this field of research and application to increase continuously in the next years [MF16].

2.2 Big Data

Sagiroglu et. al. describe Big Data as the center of modern science and business [SS13]. Big Data is initially just a term for huge data sets, not considering what the data is about. It can be generated from online transactions, emails, videos, audios, images, logs, posts, science data or, introduced in Section 2.1, the Internet of Things [SS13]. International Data Corporation (IDC) estimated a data volume of 97 zetabytes in 2022 and a increase to 181 zetabytes by 2025 worldwide[1]. Such large amount of data is not easy to handle and require special techniques for storing, processing and analyzing. The economic value of data reveals itself by an intelligent analysis to generate valuable insights [BCC20]. Secret correlations and hidden patterns need to be extracted from a heterogeneous and complex structure to make use of it.

In general, data can be structured or unstructured. The latter is random and, thus, especially difficult to analyze [SS13]. Companies resort to the cloud computing model for efficient management and analysis of Big Data [BCC20]. Bansal et. al. describe 13 challenges for dealing with Big Data from IoT environments. Among the volume, variety and velocity, already introduced in Chapter 1, they address the veracity and validity of the data. Veracity is the issue of uncertainty about truthfulness of captured data [BCC20] and validity about the cleanliness to carry out data analysis. Especially in the IoT, this is a big challenge, since a decentralized network of low power devices is error prone. Data errors and possible reasons for these are introduced in Section 2.3.

The massive amounts of data and the ease of data collection nowadays leads to an increasing interest in machine learning. Machine learning algorithms have never been more promising and also challenged by Big Data in gaining new insights into various business applications and human behaviors [ZPWV17]. Such algorithms rely on rich and large data to learn from it and to uncover underlying structures or to make predictions [ZPWV17]. That in turn calls for even faster techniques to acquire and collect data. Data augmentation is an approach to gather data from existing datasets. With the help of augmentation technologies, data can be further enriched with existing entity information [RHW21]. To be able to learn from data, supervised machine learning algorithms require them to be labeled. Annotations can be made manually, which is very cumbersome and labor intensive, as well as automated. Zhou et. al. describe several techniques to face this challenge: (a) use existing labels, (b) crowd-based or (c) use weak labels. While the first two methods try to estimate the label using machine learning or crowd-sourcing techniques, weak labels represent an alternative approach. Weak labels are not perfect but compensate for the lower quality due to their large quantities [ZPWV17].

In summary, Big Data can be the “new oil of the future” [RHW21], but there are still challenges which need to be addressed to fully take advantage of such large data sets. The “cleanliness” of data is one of them and is described in the next section.

2.3 Data Errors

In the federal standard 1037C [TD96] an error is defined as:

[1] https://www.statista.com/statistics/871513/worldwide-data-created/

Definition 2.3.1
(a) an accidental condition that causes a functional unit to fail to perform its required function, (b) a defect that causes a reproducible malfunction or (c) an unintentional or partial short circuit between two energized conductors.

Errors in data, on the other hand are defined as a condition in which data has been altered erroneously and is not equal to the ground truth anymore. Cardoso et. al. defines the ground truth as data that is related to more consensus or reliable values/aspects, so that it can be used as references [CPIR14]. The ground truth can be considered as the true and unaltered value of a specific data cell. In the IoT, data inaccuracy is introduced at the data provenance level due to several reasons such as, sensor malfunctioning, lack of trust and reliability of data sources [BCC20]. Since IoT devices are mostly interconnected in large complex networks, the probability of a failure or data error is increased compared to traditional central computing [SDSB19]. Long multi-hop journeys of data across heterogeneous networks affect data accuracy by the introduction of noisy, missing, or redundant values [BCC20]. Not at least the use of wireless communication leads to another reduction of data quality [FP05]. To accomplish the desired mobility, small size and low energy consumption for such devices, errors must be accepted. It is up to the data processing units to validate data produced by the IoT and filter erroneous cells or directly repair them. This process is called *Data Validation* and will be introduced in Chapter 4.

This thesis distinguishes between data errors in numerical cells and in textual cells. Errors in datasets with mainly text-based data are typically syntactically, but not necessarily semantically, wrong in comparison to the true value [SPKN20]. Nonetheless, real data contains domain-specific relationships beyond syntactic equivalence or similarity [ZZL+21]. Semantic errors proposes another challenge to error detection mechanisms (Chapter Chapter 4). Faults in numerical datasets are introduced in the following Section 2.3.1.

2.3.1 Numerical Value Errors

Measurements, taken from sensors in the IoT are typically numerical values. Thus, datasets recorded in IoT environments contains lots of numbers, which have to be treated differently in error detection than just comparing them syntactically. Numbers can deviate only in some low decimal places, so almost the same, but when comparing their strings, more than half of the characters are different. Chakraborty et. al. defined several types of numerical data error (Table 2.1).

Short	A data point that significantly deviates from the expected value or time.
Spikes	A rate of change over a shorter or longer period of time, that is significantly greater than expected. The change does not have to return to normal.
Stuck-at	A data series that no longer varies over a long period of time or varies very little.
Noise	Data that unexpectedly varies greatly over a certain period of time.
Calibration	Sensor data have a constant shift with respect to the true value.

Table 2.1: Subdivision of numerical value errors according to Chakraborty et al. [CNC+18]

Shorts, *Spikes*, as well as *Noise* can be grouped under the term outlier or anomaly. Hawkins defined an outlier as follows [Haw80]: "An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism." This thesis mainly focuses on outlier detection for numerical values. Chapter 4 shows, how discovering and using dependencies from IoT environments can help to improve the detection performance.

2.4 Ontologies

Ontologies are collections of expert knowledge and offer a unified semantic for the components of a specific domain [DABS22]. Since knowledge-based systems are expensive to build, test and maintain, a flexible, easy to use and expandable methodology is needed [Gru93]. In computer science, ontologies are formal models of a domain which support the understanding between humans and machines [HZA+06]. They provide a detailed insight about existing objects, structure and relationships and are suited to represent context information. In such an ontology, definitions associate the names of entities in the domain (e.g., classes, relations, functions, or other objects) with human-readable text, describing what the names are meant to denote, and formal axioms that constrain the interpretation and well-formed use of these terms [Gru93].

Ontologies are built of statements, also called triples. A triple consists of a subject, a descriptive predicate and an object [MW14]. The basic structure can be extended to support more complex models by using triples as objects or subjects of other triples. Given this format, knowledge can be represented machine-readable. The structure of an ontology is flexible, easy to scale, and can thus be generic and extendable, while allowing a good description of heterogeneous systems [DABS22].

Definition 2.4.1
Ontologies can be defined as a tuple $O = (C, HC, RC, HR, I, RI, A)$*, with concepts* C *that are arranged in a hierarchy* H *[HZA+06]. Relationships* R *between single concepts can be also arranged hierarchically* HR*. An Instance* I *is interconnected by property instances* RI*. Furthermore, axioms* A *can be defined which provide integrity constraints.*

These integrity constraints will be useful for later feature extraction for data cleaning, which is the main contribution of this thesis. In the IoT, ontologies are especially useful to represent their complex and heterogeneous structure, while still being fine-grained. This thesis uses the IoT Context Model introduced by Del Gaudio et. al. [DABS22] as an existing ontology to extend and use for automatically extracting dependencies from it.

2.5 Data Dependencies

Most data is not completely unstructured, but contains some kind of relations between the data points. Data is called dependent, if some tuples, fulfilling certain conditions, exist in the dataset. Then either some other tuples must also exist therein, or some values in the given tuples must be equal [BV81].
Two classes of dependencies, i.e. Functional Dependencies (Section 2.5.1) and its further development Ontology Functional Dependencies (Section 2.5.2) are introduced in this section.

2.5.1 Functional Dependencies

A Functional Dependency (FD) is a statement which uniquely determines the value of an attribute Y given a set of attributes X [ZGR20]. They mostly describe relationships based on syntactic equality and can be used, e.g., for data cleaning [BKC+17]. Dependencies are used to specify data quality requirements.

Definition 2.5.1
Considering data D with relation R and the FD $X \rightarrow A$, where $X \subseteq R$ denotes a set of attributes and $Y \in A$ is a single attribute, an instance I of R satisfies F if for every pair of tuples $t_1, t_2 \in I$ if $t_1[X] = t_2[X]$, then $t_1[A] = t_2[A]$. [ZZL+21]

To derive FDs from data observations, identification of the attribute order which defines the direction, is needed [ZGR20]. In order, to reduce the exponential computational cost, existing methods rely on pruning to search over the lattice of attribute combinations [ZGR20]. Different Types of FDs are (a) *Inclusion Dependencies*, (b) *Conditional Functional Dependencies* [CFG+07], (c) *Denial Dependencies* [BKC+17], (d) *Order Dependencies* or (e) *Matching Dependencies* [ZZL+21]. This thesis only focuses on denial dependencies and matching dependencies, to enhance data validation by extracting them from the structure of the data, modeled in ontologies (Chapter 4). Additionally, new types of dependencies are defined and used by the implemented concept.

The following dataset contains real data of clinical records from the Linked Clinical Trials database[2].

CC	CTRY	SYMP	DIAG	MED
US	United States	joint pain	osteoarthritis	ibuprofen
IN	India	joint pain	osteoarthritis	NSAID
CA	Canada	joint pain	osteoarthritis	naproxen
IN	Bharat	nausea	migrane	analgesic
US	America	nausea	migrane	tylenol
US	USA	nausea	migrane	acetaminophen
IN	India	chest pain	hypertension	morphine

Table 2.2: Sample clinical trials data

From the data in Table 2.2, the following Functional Dependencies can be derived [AZC+18]: F_1 : [CC] $\rightarrow$ [CTRY] and F_2 : [SYMP, DIAG] $\rightarrow$ [MED].
However, the FD F_1 is violated by the tuple (t_1, t_5, t_6), because they are not syntactically equal. A human knows that 'United States' is a synonym with 'America' or 'USA', thus t_1, t_5 and t_6 all referring to the same country. But this knowledge is not applicable for an automated process that only validates the data based on the Functional Dependencies F_1 and F_2. For this reason, Zheng et. al. introduce Ontology Functional Dependencies, which additionally consider domain knowledge [ZZL+21].

[2] https://clinicaltrials.gov/

2.5.2 Ontology Functional Dependencies

As seen in the previous section, Functional Dependencies are limited in their application for evaluating datasets due to missing domain knowledge. Ontology Functional Dependencies (OFDs) can solve those issues. They serve as contextual data quality rules that enforce the semantics modeled in an ontology [AZC+18]. Zheng et. al. focus their work with OFDs on capturing synonyms and is-a relationships, which are defined in ontologies. These relations go beyond syntactic equality or similarity and consider the notion of sense. Considering a sense or interpretation for each equivalence class $x \in \Pi_X(I)$ under which all the A-Values of tuples in x are synonyms, a relation instance I satisfies a synonym OFD $X \rightarrow_{syn} A$ [ZZL+21].

Given example in Table 2.2 above, all designations representing the same countries can be summarized in an equivalence class (e.g. US: t_1, t_5, t_6), which gives $\Pi_{CC} = \{\{t_1, t_5, t_6\}, \{t_2, t_4, t_7\}, \{t_3\}\}$. The information that links those individual names together as synonyms can be extracted, for example, from a geographical ontology, like: $names(UnitedStates) \cap names(America) \cap names(USA) =$ United States of America [ZZL+21]. This thesis extends the use of OFDs and defines new types of dependencies, which can be extracted from a ontology. Those newly defined dependencies are closely related to the IoT and are described in the main chapter, Chapter 4.

3 Related Work

In “Data Cleaning: Overview and Emerging Challenges”, Chu et. al. [CIKW16] describe detecting and repairing dirty data as one of the perennial challenges in data analytics. Qualitative error detection is crucial when acquiring large amounts of data which has influence on the decision making in businesses. Dirty data can lead to incorrect decisions and unreliable analysis [CIKW16]. Chu et. al. describe an overview over qualitative data cleaning and error repairing approaches. They conclude challenges that still need to be solved, like the scalability of data cleaning techniques or error detection applications for streaming data. This thesis introduces a concept to reduce user engagement, another challenge mentioned in [CIKW16], by automatically extracting constraints in datasets.

Zhang et. al. [ZGR20] describe the need for data constraints, like FDs, to improve data integrity and optimize queries in large datasets, but also to clean and profile data [ZGR20]. In their work “A Statistical Perspective on Discovering Functional Dependencies in Noisy Data”, they focus on extracting FDs directly from the dataset. The main goal of Zhan et. al. is to automate the discovery of those in a scalable way while the output is interpretable without any tedious fine tuning. Existing methods rely on searching over the whole space of possible attribute combinations, which can lead to poor scalability. Zhang et. al. propose that the noise in the dataset itself can be a challenge too. Due to missing values and erroneous data, the discovery method needs to extract *approximate FDs*, which are only applicable for a portion of the data. This problem leads to an even more expensive error detection problem and probably to faulty error models. In [ZGR20], they propose a framework that relies on structured learning. With the help of a probabilistic model, FDs can be discovered from noisy data. The goal of this thesis is to overcome this problem by extracting the dependencies not from the dataset itself, but from separate ontologies which will represent the expert knowledge of this domain. In future work, the two concepts can be combined to fill missing knowledge in ontologies, or on the other hand, to compensate the noise in the dataset (Section 7.1).

Cong et. al. introduce conditional functional dependencies, and a framework for data cleaning in their work “Improving Data Quality: Consistency and Accuracy” [CFG+07]. The framework is able to repair a dataset, so that it satisfies a given set of conditional functional dependencies and incrementally find a repair in response to updates to a clean database. Their approach works without incurring excessive user interaction, while still being accurate above a predefined rate. With the use of a newly defined cost model, they choose the best action to resolve a violation, which can be solved in more than one way. While Cong et. al. describe a framework for improving data quality based on conditional constraints, these constraints still have to be defined manually for every different dataset. In contrast to that, this thesis defines new constraints that can be found in datasets and furthermore methods to extract them automatically from a given ontology.

Visengeriyeva and Abedjan propose two new holistic approaches for effectively combining error detection systems in “Metadata-Driven Error Detection” [VA18]. They argue that, due to the different sources of data, multiple error detection methods are crucial to get a satisfying data quality. The structural heterogeneity of these sources is the origin of data quality problems, like missing

values, duplicates, inconsistent data and outliers [VA18]. Research in the past resulted in several data cleaning approaches but usually optimized on one specific error type. To find out which error detection method is the best fitting, knowledge about metadata is needed. Visengeriyeva et. al. consider following concepts of metadata: (a) *Data Completeness*, (b) *Data Type Affiliation*, (c) *Attribute Domain*, (d) *Frequent Values*, (e) *Multi-column Dependencies*. While (a), (b), (c) and (d) are usually considered in existing error detection frameworks, such as HoloClean [RCIR17] (see Section 5.2.1), this thesis will concentrate on dependencies in the data structure. The thesis presents concepts to extract these dependencies from ontologies which represent domain knowledge about the dataset.

In "Discovery and Contextual Data Cleaning with Ontology Functional Dependencies" [ZZL+21], Zheng et. al. explore dependency-based data cleaning with Ontology Functional Dependencies. Additionally to traditional FDs, OFDs can express semantic attribute relationships, which helps to significantly reduce the number of false positive errors in data cleaning techniques. Zheng et. al. demonstrate that real data contains domain-specific relationships beyond syntactic equivalence or similarity, for example synonyms [ZZL+21]. Such semantic models of datasets are often described in ontologies. They define OFDs based on synonym relationships and present an algorithm to discovery them from an ontology. The *OFDClean* algorithm then selects the best interpretation, so called *sense*, for an equivalence class of tuples. In datasets recorded from IoT environments, discovering semantic synonyms only plays a small role for error detection compared to text-based datasets. Since numerical values do not have a *sense* or an equivalence class, the presented concepts by Zheng et. al. are only limited for application on IoT datasets. To face this challenge, this thesis proposes multiple concepts for integrating context from ontologies in datasets consisting mainly of numerical values.

In [AZC+18], Langouri et. al. motivate a need to consider the context of data for data cleaning. They argue that the context of data can be modeled as an ontology to improve error detection steps in the data cleaning pipeline. Langouri et. al. mention that integrity constraints, e.g. FDs, are not enough to sufficiently represent relations in data and consequently will lead to an increased number of false positives in error detection. In the recent work [BKC+17], they propose an algorithm that discovers OFDs directly from data without a need to model the context. Additionally, two open problems are summarized in [AZC+18]: (a) a concept on how to suggest possible modifications of data that violate a given set of contextual dependencies and (b) holistic data cleaning algorithms that consider data, dependency and ontology repairs. These problems are partly solved by a data validation framework HoloClean [RCIR17], which will be introduced in the following. The goal of this thesis is to present a concept for (b), mentioned as future work by Langouri et. al.. New types of dependencies, augmented with ontological context and develop are defined (Section 4.4.3), and through implementing a prototype, a concept on how to discover and use them for error detection, is presented (Chapter 5).

HoloClean [RCIR17] is a framework for holistic data repairing driven by probabilistic inference. It unifies existing qualitative data repairing approaches, which rely on integrity constraints or external data sources, with quantitative data repairing methods, which leverage statistical properties of the input data. The framework generates a probabilistic model to represent the input data. With the use of statistical learning and probabilistic inference over the generated model, HoloClean repairs errors in the input data. Rekatsinas et. al. show that their data cleaning method outperforms state-of-the-art alternatives by the factor 2 in F1-score. HoloClean finds repairs with average precision of 90% and an average recall of 76%. They limit their approach to *Denial Constraints* and *Matching Dependencies* (Section 2.5.1). In this thesis, the methods of HoloClean are extended

to extract these dependencies from an ontology and, at the same time, use newly introduced data dependencies for error detection.

Mahdavi et. al. present a new configuration-free error detection system with Raha [MAF+19]. Detecting erroneous values currently require a user to provide configuration for various error detection methods, e.g. outlier detection. They argue that this selection is not trivial and needs individual adjustments for every new dataset. Error detection in general requires users to engage these non-trivial steps: (a) algorithms selection, (b) algorithm configuration and (c) result verification. With their implementation Raha, they try to solve all of this time-consuming steps in an automated manner. The algorithm automatically generates different configuration sets for each error detection method. With a novel sampling and classification scheme for generated feature vectors, Raha chooses the most representative values for training. Although the goal of Raha and this thesis is the same, namely to have as little user involvement in the data cleaning process as possible, this thesis uses expert knowledge from ontologies and does not rely on structures which firstly needs to be discovered in the dataset.

Del Gaudio et. al. evaluate different ontologies, which form the base of their work, and proposes an expandable architecture for a context model using the IoT-Lite ontology [BEBT16] and the Semantic Sensor Network (SSN) ontology [CBB+12]. The IoT-Lite ontology uses current standards and is compatible to be extended with additions from different sources without any constraints [DABS22]. Del Gaudio et. al integrate dynamic context data, such as measurements, timestamps and monitoring components to the ontology. They use a so-called *Context Model Store*, which represents a centralized database, that stores attributes and predicates, describing the context. Additions to the ontology can be made during runtime via the *Context API* [DABS22]. The IoT environment can be automatically parsed to an ontology through querying the Multi-purpose Binding and Provisioning Platform [FHM19; FHPM18]. The MBP is introduced by Franco da Silva et. al. as a management platform for IoT environments. Devices, sensors and actors can be registered, controlled and redeployed from a central system. The acquired data can be processed and visualized after the required operators are deployed and new data is collected [DABS22]. The ability of the ontology, to be automatically generated from existing IoT platform tools, decreases human involvement significantly. Therefore, the proposed IoT context model by Del Gaudio et. al. forms the base of the context model presented in this thesis.

4 Context-Aware Data Validation

The main concept of this thesis, the context-aware data validation, is introduced in this chapter. In the beginning, general data validation and errors are introduced (Section 4.1). Afterwards, the overall architectural structure and the data flow of the concept, is presented (Section 4.2). The goal is to improve data validation through considering constraints and dependencies gathered from ontologies about the same context as the dataset. Therefore, the environment where data is generated needs to be modeled. For this concept, I use the IoT context model ontology by Del Gaudio et. al. [DABS22]. The ontology is extended with additional classes and properties (Section 4.3). Ihe extensions are illustrated with an example for such a context model in Section 4.3.4. New types of OFDs are introduced (Section 4.4) that can be extract from a context model (Section 4.5). Lastly, I conceptually describe how to use them for improving the data validation process automatically (Section 4.6).

4.1 Definition and Overview of Data Validation

The UNECE glossary on statistical data editing[1] defines data validation as follows: "An activity aimed at verifying whether the value of a data item comes from the given (finite or infinite) set of acceptables values." So either the value is "acceptable" or can be interpreted as an error (Section 2.3). A data error can formally be defined as follows [MAF+19]:

Definition 4.1.1
Let a dataset D consists of tuples t_k. Let A be the schema of D with attributes a_l. Then $D[i, j]$ is the data cell in the tuple t_i of dataset D and a_j the attribute of the schema A. One can denote the ground truth of the same dataset as D^*. An data error is then every data cell $D[i, j]$ which is different from the corresponding cell in D^*, $D^*[i, j]$. So if $D[i, j] \neq D^*[i, j]$, $D[i, j]$ is considered erroneous.

The definition of the UNECE glossary on statistical data editing only considers single data items, without mentioning the verification of consistency among different data items [ZFG+16]. Therefore, Di Zio et. al. propose a new definition for data validation: "Data Validation is an activity verifying whether or not a combination of values is member of a set of acceptable combinations." To validate the combination of values, any type of context of the data is needed. Most current state-of-the-art techniques uses rules or constraints to evaluate the correctness of data points [BKC+17; CFG+07; VA18; ZFG+16]. Chu et. al. mention in their overview over data cleaning, how time-consuming it is to design such constraints or patterns manually. Later, in the evaluation chapter of this thesis (Section 6.1.1), one can see how crucial it is that the set of constraints needs to be the most complete

[1] https://unece.org/info/Statistics/pub/21882

as possible. Automatic discovery techniques are essential, due to the requirement of great domain expertise in this process [CIKW16]. As seen in previous works about challenges in data validation, meta data, representing constraints and dependencies, is important to consider [CIKW16].

4.2 Concept Overview

The main objective of this thesis is to enrich existing data cleaning methods with features extracted from an ontology-based context model. The generation of these features and the resulting data validation should be done with least human involvement possible. Therefore, a data cleaning pipeline is proposed which divides the process into three parts: (1) data generation and context modeling, (2) OFD extraction and feature generation, and (3) data cleaning. These three steps are also reflected in the architectural overview in Figure 4.1.

This section describes the concept superficially and should give an rough idea, while the following sections describe the individual components more in detail.

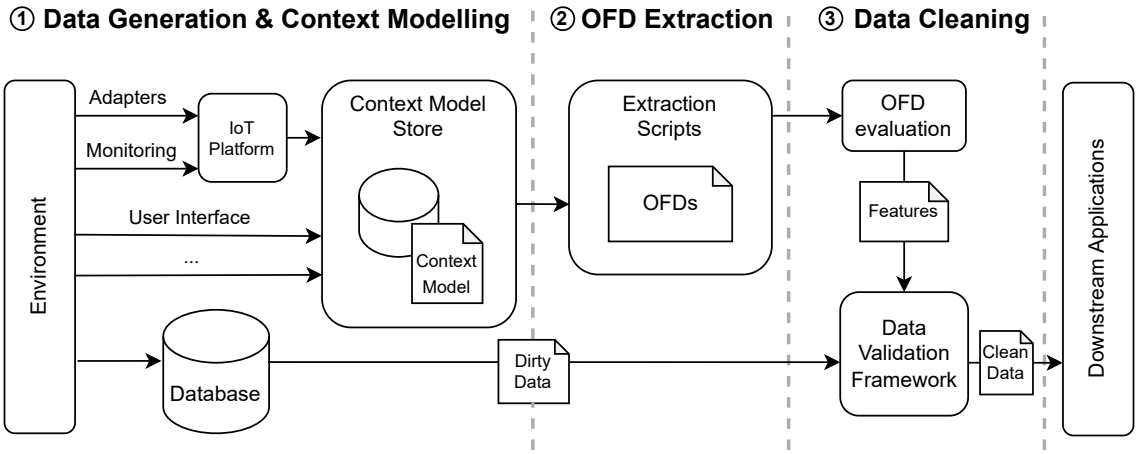

Figure 4.1: Architectural overview over the proposed data cleaning pipeline

The environment produces raw data that is stored in a database, for offline validation, or directly streamed into the data validation framework. Del Gaudio et. al. offer various adapters in their work to extract information about the environment [DABS22]. This information can be modeled automatically in an ontology-based context model. Modern IoT platform tools, e.g., the MBP presented by Franco da Silva et. al. [FHM19; FHPM18], can deliver such information via application interfaces. Otherwise, the context model can be created manually, e.g., with a user interface. In principle, the context model can be created in any way and inserted into the presented pipeline. In the following section, this model is described more in detail, along with challenges when creating such.
The context model is then parsed and available dependencies are extracted from it. These OFDs (Section 2.5.2) can be categorized in different types, which will be presented in Section 4.4. When validating the incoming data, the extracted dependencies are evaluated against it. If a dependency holds on a given data tuple, this tuple is considered correct, while if a dependency is violated, the

tuple is marked as erroneous. This boolean feature can then be further processed in an arbitrary data validation pipeline, for example in ready-to-use frameworks like HoloClean [RCIR17] or Raha [MAF+19]. With this addition, any already implemented error detection and repair mechanism can still be used.
Therefore, the presented architecture in this thesis allows for the combination of already existing techniques in data validation and the use of automatically extracted dependencies from context models to improve the results. At the end of the presented pipeline, the cleaned data will be outputted and can be used in other downstream applications, e.g., machine learning applications and data visualization dashboards.

4.3 Step 1: Data Generation and Context Modeling

Data from the environment can either be processed directly as a stream or inputted as a static dataset. Such data can, e.g., be generated by sensors, which measure some physical value, be the output of some kind of data processing unit, or be a tabular collection of data about a specific topic. Since data normally is not produced randomly, in the most datasets, constraints or correlations between rows or cells exist [ZFG+16]. To automatically make use of such, the information, e.g., the structure of an IoT environment, needs to be embedded in a model first.
Knowledge about the environment is not only important in IoT datasets, but also can be helpful in datasets from other domains, yet, it is important to notice that this thesis mainly focuses on the validation of IoT data. Due to the increasing complexity of data analysis and the increasing number of data producing devices, more detailed context models are needed, which not only cover the static information about the network, but also compose live information, for example the current state of devices or live data from monitoring systems [DABS22].

As mentioned in the previous section, the goal of this thesis is to use this information to improve data validation. For this concept, ontologies are used as a knowledge collection. Due to their flexibility and extendability, ontologies are very applicable for this task, as mentioned in Section 2.4. They are easy to understand, can be maintained automatically and easily extended whenever needed. In the following, the context model is introduced, which builds on the work of Del Gaudio et. al. [DABS22]. The ability to be generated from the Multi-purpose Binding and Provisioning Platform (MBP)[2], a platform tool to manage and monitor IoT devices [FHM19; FHPM18], gives the IoT context model an advantage, due to the low involvement of humans.

4.3.1 IoT Context Model

The presented context model in this thesis builds on the IoT context model by Del Gaudio et. al.. In the following, extensions to this ontology are described that enable modeling of new dependencies in the environment. The context model is extended with components from the SSN ontology and the IoT-lite ontology, introduced in the following. The ontologies use current standards and are compatible to be extended with additions from different sources without any constraints [DABS22]. Del Gaudio et. al integrate dynamic context data, such as measurements, timestamps and monitoring

[2] https://github.com/IPVS-AS/MBP/blob/master/README.md

components to an ontology. The context-aware data validation method of this thesis uses the context model store as introduced by Del Gaudio et. al. [DABS22]. The context model store is a centralized database, that stores attributes and predicates, describing the context and can be changed during runtime. An IoT environment can automatically be parsed to an ontology through querying an IoT platform, e.g., the Multi-purpose Binding and Provisioning Platform [FHM19; FHPM18]. The MBP is introduced by Franco da Silva et. al. as a management platform for IoT environments. Devices, sensors and actors can be registered, controlled and redeployed from a central system. The acquired data can be processed and visualized after the required operators are deployed and new data is collected [DABS22]. Information, like the name, IP address, or the type of the device is represented in the context model as soon as a device is registered. Sensors and actuators can be equipped with software components, called extraction or control operators [FHM19]. These operators generate non-static data that further describes the context of the environment, but can change constantly and needs to be represented dynamically in the resulting context model [DABS22]. When the whole system is initiated, changes in the environment are constantly registered using different adapters. Thus, if a stream of data is originating from the environment, the process of data cleaning can timely be adapted to the current context.

Figure 4.2 depicts an example for the IoT context model [DABS22]. The ontology does not only include static information about the environment (grey), but also non-static information (blue), like monitoring operators. The property `iot-context:hasMonitoringComponent` connects devices with their corresponding monitoring component, which in turn links with `iot-context:hasMeasurement` to a specific recorded value (`iot-context:hasValue`) at a timestamp (`iot-context:hasTimeStamp`) [DABS22].
It is important to mention, that the context model can be generated by any platform, representing the environment. This thesis uses the MBP due to the already implemented adapters and the automated processes to construct an ontology from an environment, introduced by Del Gaudio et.al.. Furthermore, it is even possible to manually create an ontology, which is then used in the data cleaning pipeline, presented in this thesis. The concept is extendable to make use of other methods, e.g. defining the context via a user interface, as well.

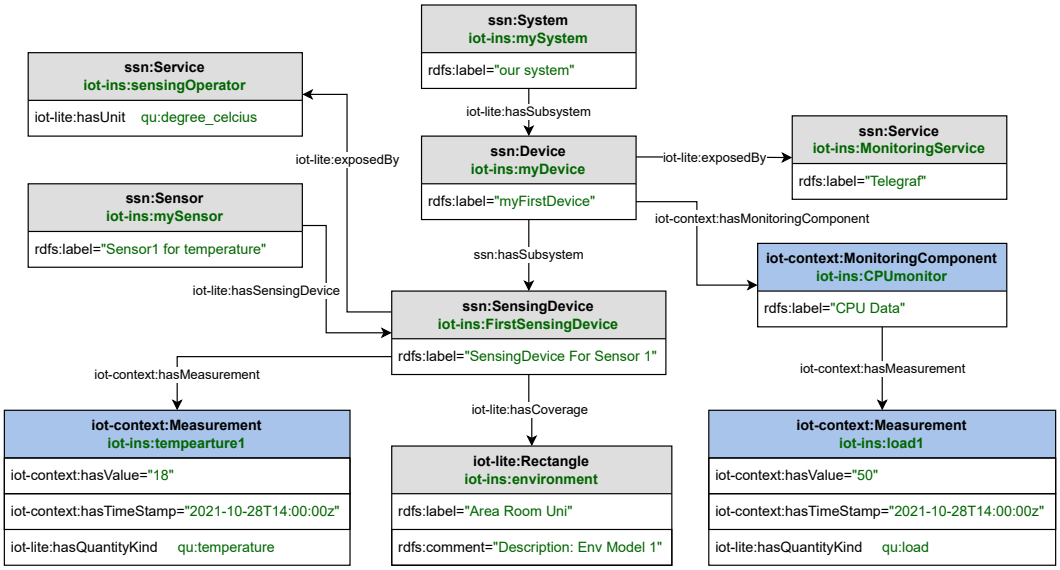

Figure 4.2: Example of the IoT Context Model [DABS22]

The IoT context model already includes useful relations for data validation purposes (Figure 4.2). Still, an IoT environment contains much more information, which will be needed to further improve error detection. Thus the existing context model is extended with additional classes and properties from other, commonly used, ontologies, as described in the following. Either these extensions are added manually to the context model in an offline phase or can be automated in future work.

4.3.2 Adding the Semantic Sensor Network Ontology

The Semantic Sensor Network (SSN) ontology [CBB+12] was introduced by the World Wide Web Consortium (W3C) to only describe sensors, observations and related concepts[3]. Compton et. al. designate the components to be the core of empirical science. Sensors are used in a variety of applications and therefore the most important part to gather an increased volume of data, which in turn is crucial for research [CBB+12]. Other information about the environment, e.g units of measurements, locations, or sensor types, should be included from other ontologies and is not part of the SSN ontology. Compton et. al. describes the SSN ontology as a system perspective, with a focus on systems of sensors and deployments [CBB+12]. Nonetheless this basic information about the sensors and their deployment is important for modeling the context of an IoT environment, which mostly consists of devices like such and attached sensors and actuators. The classes and relations, which are listed in Table 4.1, will be used by this work to extend the IoT context model. Some of the classes are already used by the IoT context model, but are still listed below for completeness.

type	**name**
class	`ssn:Device`
class	`ssn:Sensor`
class	`ssn:SensingDevice`
class	`ssn:ActuatingDevice`
relation	`ssn:hasDeployment`
relation	`ssn:hasSubsystem`
class	`ssn:Deployment`

Table 4.1: Additionally used classes and relations from SSN ontology

The relation `hasSubsystem` between the classes `Device` and `Sensor` can model existing links between sensors and devices. Thereby, it is clear if a sensor physically belongs to a device, i.e. the device can directly read values from it. `SensingDevice` and `ActuatingDevices` describe a smart device that implements some kind of sensing using a sensor or acting using an actor, respectively. The `Deployment` of a `SensingDevice` models the process of the device being deployed for a particular purpose. In this thesis, the relation `hasDeployment` referring to the class `Deployment` is mainly use to specify the location of a device in the physical environment.

[3] https://www.w3.org/2005/Incubator/ssn/ssnx/ssn

4.3.3 Adding the IoT-lite Ontology

To represent resources, entities and services in IoT environments, the W3C introduced the IoT-lite ontology [BEBT16][4]. IoT-lite is a instantiation of the SSN ontology (Section 4.3.2). The ontology is built to be queried without consuming excessive processing time, which is advantageous when using it in error detection in nearly real-time systems. Since IoT devices are mostly connected wireless and rely on battery power, the topology of an IoT environment can change often and rapidly [SDSB19]. Thus, IoT-lite considers the constrains and dynamicity of IoT environments [BEBT15]. Measurements by IoT devices can be described, using common quantity taxonomies, for example qu[5]. This ontology describes the conceptual model for quantities, units, dimensions or values. With this, measured values can be annotated, e.g., with a unit. IoT-lite focuses on the most used concepts for data analytics in IoT applications, such as sensory data, location and type [BEBT16]. It avoids links to uncommonly used ontologies and shares vocabulary with other data from different sources. All in all, IoT-lite is very suitable for the purposes of this thesis, while still keeping standards and its extensibility for future work.

type	**name**
property	`iot-lite:hasSensingDevice`
individual	`iot-lite:Attribute`
property	`iot-lite:isActed`
property	`iot-lite:isAssociatedWith`
individual	`iot-lite:Object`
property	`iot-lite:hasMetadata`
individual	`iot-lite:Metadata`
data property	`iot-lite:metadataType`
data property	`iot-lite:metadataValue`
property	`geo:hasLocation`[1]
individual	`qu:QuantityKind`[2]
individual	`qu:Unit`[2]
property	`qu:hasQuantityKind`[2]
property	`qu:hasUnit`[2]

[1] geo namespace for location property of sensors (https://www.w3.org/2003/01/geo/)
[2] qu namespace for units and quantity kinds of measurement (https://www.w3.org/2005/Incubator/ssn/wiki/QU_Ontology)

Table 4.2: Additionally used individuals, properties and data properties from IoT-lite ontology

Table 4.2 lists all individuals and properties needed for the dependencies used in data validation (Section 4.4), introduced by this thesis. The property `hasSensingDevice` links sensors to their corresponding `ssn:SensingDevice`, i.e. their directly connected device. A sensor is annotated with a quantity kind, e.g. temperature or humidity, which corresponds to the physical measurement the sensor can take. Additionally, the units that the sensor is capable to read are modeled via the property `qu:hasUnit` and the individual `qu:Unit`. Capabilities of sensors, e.g. the resolution or the minimum/maximum measurable value, can be modeled via the individual `Metadata` and its data

[4] https://www.w3.org/Submission/2015/SUBM-iot-lite-20151126/
[5] https://www.w3.org/2005/Incubator/ssn/wiki/QU_Ontology

properties `metadataType` and `metadataValue`. The individual `Attribute` defines the name of a data entry of some measurement and is associated with the property `isAssociatedWith` to a sensing device. A link to an `ActuatingDevice` can be modeled with the property `isActed`, which means that the actor does something, e.g. process or log a measurement, after it is transmitted from a sensor. With this linkage temporal relations between devices can be modeled and expressed.

An example of the IoT context model with the additional classes and properties from the other ontologies (SSN and IoT-lite) is presented in Figure 4.3 in the next section.

4.3.4 Example of the Context Model

In this section, an IoT dataset is introduced to illustrate the extension with the new classes and relations from the IoT-lite and the SSN ontologies. Moreover, the example is useful for the following definitions of the various OFDs in Section 4.4. Table 4.3 shows an excerpt of the data collected

Device	SensingDevice	Sensor	name	value	timestamp	location
device_out	aqara_multisensor_2	aqara_temp_2	t4	7.08	2021-01-01 02:00:00	outside
device_in_1	esp8266_2	ds18b20_2	t2	23.69	2021-01-01 02:00:00	room1
device_in_1	esp8266_1	ds18b20_1	t1	22.69	2021-01-01 02:00:00	room1
device_in_2	aqara_multisensor_1	aqara_temp_1	t3	22.32	2021-01-01 02:00:00	room2
device_main	raspberry		processor	7.08	2021-01-01 02:00:05	outside
device_main	raspberry		processor	23.69	2021-01-01 02:00:05	room1
device_main	raspberry		processor	22.69	2021-01-01 02:00:05	room1
device_main	raspberry		processor	22.32	2021-01-01 02:00:05	room2
device_in_1	esp8266_1	ds18b20_1	t1	22.46	2021-01-01 03:00:00	room1
device_out	aqara_multisensor_2	aqara_temp_2	t4	7.25	2021-01-01 03:00:00	outside
...	...	...	...	...	...	...

Table 4.3: Excerpt of the IoT dataset

in a smart-home IoT environment (Figure 5.2). Along with temperature read-outs from different kinds of sensors, it contains data points, logged by a central processing unit, where all of the measurements are collected. More details to this dataset can be found in the description of the implemented prototype in Section 5.1.

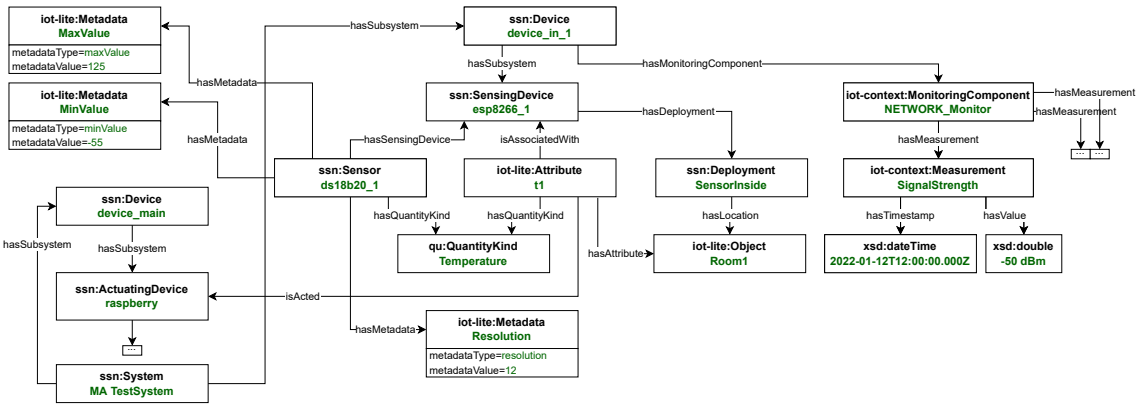

Figure 4.3: Excerpt of IoT context model for a given environment (Section 5.1.3)

In Figure 4.3, a part of the corresponding ontology is depicted. This instance of the IoT context model represents the structure of the environment, in which the dataset of Table 4.3 was collected. In the shown cutout, two devices, an *ESP8266*[6] and a *Raspberry PI*[7], are part of the overall system. One is an actor which is acted on behalf of the other device, a sensor. The sensor measures the temperature in *Room1* and is monitored via a monitoring component. It is annotated with several capabilities, e.g., the maximum and minimum measurable value and its measuring resolution. This IoT environment is used to exemplify the following definitions of the new types of OFDs.

4.4 Dependencies in the Ontology

This section defines new types of dependencies in ontologies, namely OFDs (Section 2.5.2), which are used to enhance data validation. These types are categorized into three main categories: (a) structure-based, (b) time-based, and (c) value-based. Structure-based OFDs represent the structural relations of a dataset. They are especially useful in datasets consisting of mostly categorical values, since the other types of OFDs are only restrictively applicable, e.g. value-based. The latter, and time-based OFDs are mainly defined to represent the structure of an IoT environment. Examples for every dependency are given in the evaluation chapter (Chapter 6), after the corresponding datasets are introduced.

4.4.1 Structure-based Dependencies

In the following, only dependencies are defined, which take the structural relations of datasets into account. The concept, presented in this thesis, will include denial dependencies and matching dependencies. Such dependencies are applicable to many datasets since the actual cell values do not matter. Thus, structure-based constraints can be found in datasets with mostly text, as well as in datasets containing mostly numerical values. Nonetheless, in the latter, their usefulness can be limited. This is the consequence of structural-based constraints using syntactical or semantical equivalence to determine the correctness of a cell. Numbers can deviate only in some low decimal places, so almost the same, but when comparing their strings, more than half of the characters are different. Due to those restrictions and their capabilities only to cover general rules in the structure, the use of such dependencies are limited in numerical datasets, like most of the data produced by IoT. Denial dependencies and matching dependencies can be used in approaches for more general datasets. For illustrating these structure-based OFDs, excerpts of a ontology are depicted, which model a dataset that mainly consists of text-based cells with categorical values. The dataset and the corresponding ontology is introduced in Section 5.1.1. To enhance data validation especially in IoT datasets, this thesis introduces *device link dependencies*.

[6] https://www.espressif.com/en/products/socs/esp8266

[7] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

Denial Dependency

Denial dependencies subsume several types of integrity constraints, for example Functional Dependencies (Section 2.5.1) or conditional functional dependencies [RCIR17]. They indicate if a pair of cells in a dataset is faulty, i.e., if the constraint is satisfied. A denial dependency D over a relation R is defined as $A \rightarrow B$, where A and B are single attributes in R and $A \neq B$. An instance I satisfies D if for every pair of tuple $t_1, t_2 \in I$, if $t_1[A] = t_2[A]$, then $t_1[B] \neq t_2[B]$. The elements of the instance can then be marked as erroneous.
Denial dependencies are *transitive* but not *symmetric*.

Proof for transitivity:

Let $A \rightarrow B, B \rightarrow C$ be a denial dependency, then A is a subclass of B and B is a subclass of C. Furthermore A is also a subclass of C, which means $A \rightarrow C$ is a valid denial dependency ($A \rightarrow B \wedge B \rightarrow C \Rightarrow A \rightarrow C$).

Proof for non-symmetry:

Let $A \rightarrow B$ and $B \rightarrow A$ be denial dependencies. Then A is a subclass of B and B a subclass of A. This requires $A = B$, but contradicts the assumption $A \neq B$ of a denial dependency ($A \rightarrow B \nRightarrow B \rightarrow A$).

Subclass relations in ontologies have the same properties like denial dependencies, so they can be discovered directly from the class hierarchy.

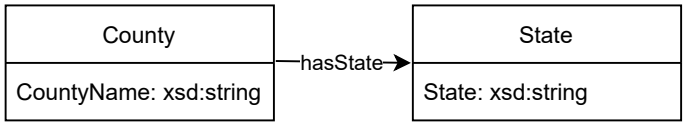

Figure 4.4: Example for parent class and subclass relationship

Figure 4.4 depicts the class - subclass relation between *County* and *State*. *County* is represented as a subclass of the class *State* in the hospital ontology (Section 5.1.2). This relation can be converted to a denial dependency *County* $\rightarrow$ *State*. Let $t_1[A] = t_2[A]$ be the county *Lassen County*, $t_1[B]$ the state *California*, and $t_2[B]$ the state *Nevada*. Then the tuple t_1, t_2 satisfies the denial dependency, thus $I = t_1, t_2$ can be considered as errors.

Matching Dependency

A matching dependency M over a relation R is represented as $A \rightarrow B$, where A and B are single attributes in R and $A \neq B$. They indicate if a pair of cells in a dataset is correct, i.e., if the constraint is satisfied. An instance I satisfies D if for every pair of tuple $t_1, t_2 \in I$, if $t_1[A] \approx t_2[A]$ and $t_1[B] \approx t_2[B]$ [SPKN20]. $\approx$ denotes the similarity operator w.r.t some similarity metric [SPKN20].
Matching dependencies are *transitive* and *symmetric*.
In the following, mathematical proofs for matching dependencies in the structure of an ontology are defined.

Proof for transitivity:

Let Φ be a general function $X \rightarrow Y$, which maps an attribute in an ontology X to its corresponding class Y. $\Phi(X)$ then denotes the class of attribute X. Let $A \rightarrow B, B \rightarrow C$ be a matching dependency, then A, B and B, C are attributes of the same class, $\Phi(A) = \Phi(B)$ and $\Phi(B) = \Phi(C)$. Since the assumption holds that every attribute is unique to one class, the class of A, B, C must be the same, $\Phi(A) = \Phi(B) = \Phi(C)$ This means that $A \rightarrow C$ is also a valid matching dependency, because they are attributes of the same class ($A \rightarrow B \wedge B \rightarrow C \Rightarrow A \rightarrow C$).

Proof for symmetry:

Consider Φ as a function defined above. Let $A \rightarrow B$ be a dependency, then A and B are attributes of the same class $\Phi(A) = \Phi(B)$. This proves that the matching dependency $B \rightarrow A$ is valid, thus symmetric ($A \rightarrow B \Rightarrow B \rightarrow A$).

Every unique attribute of a class in a ontology is dependent to each other, regarding matching dependencies.

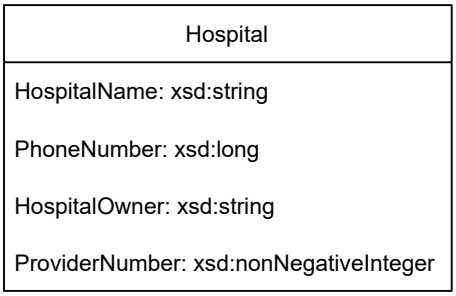

Figure 4.5: Example for a class and its attributes

Figure 4.4 depicts the class hospital with all its attributes. Since all of theses attributes are unique in the hospital dataset, they relate with each other and fulfill the definition of a matching dependency. Let $t_1[A] = t_2[A]$ be the phone number *+1 1234 456789*, $t_1[B] = t_2[B]$ the hospital name *Callahan Eye Foundation Hospital*. Then the tuple t_1, t_2 satisfies the matching dependency, thus can be considered as correct.

Device Link Dependency

A device link dependency L can be represented as a function $\Psi : X \rightarrow Y$, where X denotes the set of sensors available in the IoT environment and Y denotes devices. An dependency $A \rightarrow B$ ($\Psi(A) = B$) is satisfied, if the sensor A is physically connected to the device B. This relation means that the sensor itself belongs to this device and can only be read out from it specifically.

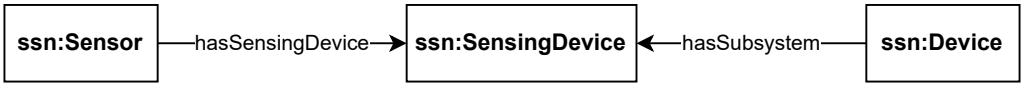

Figure 4.6: Example for a device link in an IoT ontology

In Figure 4.6, an excerpt of an IoT context model is given. The physical sensor can be mapped to a device via the link to the sensing device. Let A be a sensor, B its correspondingly connected device and t_1 a data tuple, where $t_1[sensor] = C$ and $t_1[device] = B$. Due to the dependency, $\Psi(A) = B$ must hold, but in the given example $\Psi(A) = C$. This contradicts the assumption that Ψ is a general function. Therefore t_1 can be considered as incorrect.
In the IoT environment, depicted in Figure 4.3, the sensor `ds18b20_1` is linked to the device `device_in_1`. Consequently, $\Psi($`ds18b20_1`$)$ = `device_in_1` is defined by the device link dependency that is present in the example ontology.

4.4.2 Time-based Dependencies

Time-based dependencies can be used to evaluate the plausibility, and detect errors in timestamps or wrongly ordered data. They provide constraints typically in IoT environments and represents the direction how the data is transmitted. These constraints can be discovered from the extended IoT context model as well. In the following, the *temporal dependency* is defined.

Temporal Dependency

Temporal dependencies T define a relation between two devices, A and B. If the dependency $A \rightarrow B$ holds, data, e.g., measurements, will only be send from device A to B. Let the timestamp t_X denotes the time where a message is processed on some device X. Let's consider two devices, A and B. Since transmission time is greater than 0, the timestamp t_A of message m will then be smaller than timestamp t_B of that same message, $t_A < t_B$. A can then be called a temporal predecessor of B. Temporal dependencies are *transitive*.

Proof for transitivity:

> Let $A \rightarrow B, B \rightarrow C$ be a temporal dependency, then data only flows from device A to B and from device B to C. Since receiving and processing time of a message > 0, for messages sent from A to C over B the following always holds: $t_A < t_B < t_C$. This means $A \rightarrow C$ is a valid temporal dependency ($A \rightarrow B \wedge B \rightarrow C \Rightarrow A \rightarrow C$).

Temporal dependencies are represented in the IoT context model as follows:

Figure 4.7: Example for a temporal dependency in an IoT ontology

Figure 4.7 depicts a cutout of an ontology where a specific attribute, e.g. a measurement, created by a device A is trigger for a action taken by device B. This means the temporal dependency $A \rightarrow B$ holds, since the measurement first needs to be taken before processing an action from it, $t_A < t_B$. In the IoT environment, depicted in Figure 4.3, the actor `device_main` is acted on behalf of the attribute `t1`. This attribute is measured by the sensing device of `device_in_1`. Therefore the measurement of the sensor must be temporally taken before `device_main` can process this value to do anything, e.g., log it to a database. Consequently, a temporal dependency exists between those devices, namely `device_in_1` $\rightarrow$ `device_main`.

4.4.3 Value-based Dependencies

Especially for datasets containing mostly numerical values, e.g., IoT datasets, structural constraints are limited. They can not cover a relation between multiple measurements values and detect false values reliably. Therefore, this thesis introduces value-based dependencies. Such dependencies are linking between values of different measurements or measurements and the theoretical limits of the devices, where they are measured. In the following, the *locality dependency*, the *monitoring dependency* and the *capability dependency* are introduced and defined.

Locality Dependency

A locality dependency L is defined as a function $\Gamma : Device \rightarrow Locality$. Let A be a sensing device and B a locality, e.g. a room. If the device A is placed at the location of B, then the locality dependency $A \rightarrow B$ holds and $\Gamma(A) = B$. This means, that measurements taken from the sensing device A will always capture the environment at the location B.

Figure 4.8: Example for a locality dependency in an IoT ontology

In Figure 4.8, a locality dependency is represented as a excerpt from an IoT ontology. This cutout describes a sensing device deployed in a specific location, where the measurements are taken from. With this information, the measurements from devices which are placed nearby each other, e.g., in

the same room, can be used to improve outlier detection. Values recorded in a local area are likely to correlate in their absolute values and changing rate. This helps especially in borderline situations, where it is not clear whether a normal measurement should be considered as an outlier or vice versa. Since this situation is still a common problem in outlier detection nowadays [WBH19], this thesis is going to evaluate the usefulness of considering locality dependencies in the detection process (Section 6.2.2).
The IoT environment, depicted in Figure 4.3, contains the sensing device `esp8266_1`. This device is deployed into the `Room1`, therefore can only measure temperatures from this location. $\Gamma($`esp8226_1`$)$ = `Room1` is locality dependency that can be extracted from the excerpt in Figure 4.3.

Monitoring Dependency

A monitoring dependency M describes a device A, which is monitored by a monitoring component B. Health indicators, like the CPU load or network connectivity, are stored as measurements in the IoT Context Model [DABS22]. Since those measurements are being added live while operating, monitoring dependencies can be used in real-time or on a existing dataset with timestamps.

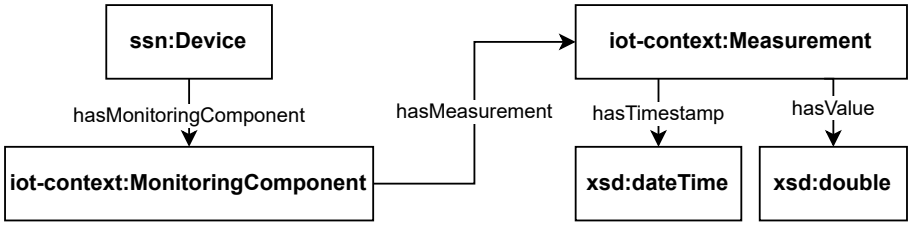

Figure 4.9: Example for a monitoring dependency in an IoT ontology

Figure 4.9 depicts a device which is monitored by a monitoring component. This component stores monitoring measurements, i.e., the measured value combined with the timestamp. A health indicator, which will be useful to monitor for a wireless device, is ,for example, the signal strength. If not using any lower level error detection protocol, e.g., Transmission Control Protocol (TCP), low signal strength can lead to higher probability of false received values [BV98]. Therefore, the information lying in monitoring measurements is useful to further improve error detection.
In the IoT environment, illustrated in Figure 4.3, `device_1` has a monitoring component. Thus the monitoring dependency holds between the device `device_1` and its corresponding monitor `NETWORK_Monitor`.

Capability Dependency

A capability dependency C describes a set of capabilities B assigned to a sensor A. One capability is represented as a metadata object for a specific sensor. The metadata object consists of a type, e.g., resolution or minimal measurable value, and its value.

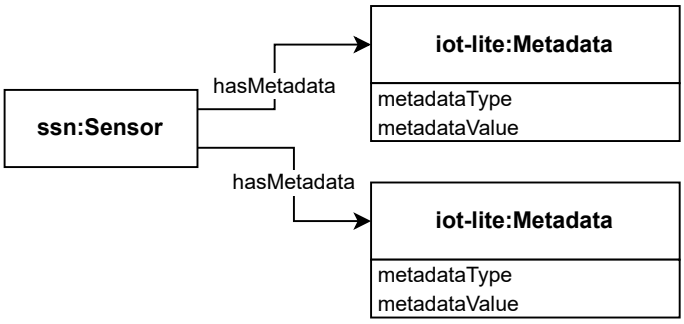

Figure 4.10: Example for a capability dependency in an IoT ontology

Figure 4.10 represents a cutout from the IoT Context Model where a sensor is linked to two metadata objects. Because metadata objects store the capabilities of a sensors, they can be used as filters for the measured values. Values which are not plausible, regarding the sensors abilities, can then directly be marked as false in error detection. For example, if a measurement is lower than the minimal measurable value of a sensor, this is considered as erroneous. A capability can also define borders of the measured units. Temperatures which are lower then the absolute zero are then automatically labeled as an error.
The sensor `ds18b20_1` in the IoT environment, depicted in Figure 4.3, has several meta datas linked to it. For example, a `MaxValue` and `MinValue`, as well as a `Resolution` of the sensor is defined. Therefore, these three capabilities are assigned to the sensor `ds18b20_1` and are interpreted as capability dependencies.

4.5 Step 2: OFD Extraction

The second step of the data validation pipeline, illustrated in Figure 4.1 is the OFD extraction. The context model is now filled with information about the environment from which the data is generated. Before the OFD evaluation and data validation step, the dependencies in the dataset, i.e. the OFDs, need to be extracted. This thesis expresses the queries of the individual OFD with the syntax of SPARQL-queries. SPARQL[8] is a query language for Resource Description Framework (RDF). It supports diverse data sources in RDF format and can query graph patterns along with their conjunctions and disjunctions. RDF is a data format which represents a directed, labeled graph. It is used mainly for representing information in the web and storing ontologies. W3C initiated a RDF Data Access working group which has identified use cases and requirements[9]. They define a standard way to query or access data in RDF format. The queries are designed to match entries in the data store, i.e., in the ontology, which is a set of subject-predicate-object triples (Section 2.4). The `WHERE` clause specifies conditions that have to apply on the queried triples. Therefore parameters can be defined, which are marked with the prefix `?`. They can be selected in the `SELECT` clause to have their binding returned.

In the following, the queries of the OFDs are introduced, structured by their types.

[8] https://www.w3.org/TR/rdf-sparql-query/
[9] https://www.w3.org/TR/rdf-dawg-uc/

Denial Dependencies

Listing 4.1 shows the SPARQL-query for extracting denial dependencies from the context model.

```
SELECT ?className ?subclassName
WHERE {{
    ?class a owl:Class .
    ?class rdfs:label ?className .
    ?class rdfs:subClassOf ?subclass .
    ?subclass owl:onClass ?name .
    ?name rdfs:label ?subclassName
}}
```

Listing 4.1: SPARQL-query for extracting denial dependencies

`className` returns the name of a class that directly inherits to a subclass, called `subclassName`. This query results in pairs of classes and all their available sub classes, which then are used to build *denial dependencies* accordingly to the description in Section 4.4.1.

Matching Dependencies

Listing 4.2 shows the SPARQL-query for extracting matching dependencies from the context model.

```
SELECT ?className
WHERE {{
    ?class a owl:Class .
    ?class rdfs:label ?className .
    ?class {0}:{1} ?property
}}
```

Listing 4.2: SPARQL-query for extracting matching dependencies

`className` returns the class of a specified property. The namespace of the property and the property itself is added as an argument to the query. For every found class that has more than one unique property, matching dependencies are created between those attributes (see Section 4.4.1).

Device Link Dependencies

Listing 4.3 shows the SPARQL-query for extracting device link dependencies from the context model.

```
SELECT *
WHERE {{
    ?device rdfs:label '{0}' .
    ?device ssn:hasSubsystem ?sensing .
    ?sensor ssn:hasSensingDevice ?sensing .
    ?sensor rdfs:label '{1}'
}}
```

Listing 4.3: SPARQL-query for extracting link dependencies

This query returns a result row if a device link exists between the sensor and the device, which are both passed by argument. If the result is not empty, a device link dependency (Section 4.4.1) is created.

Temporal Dependencies

Listing 4.4 shows the SPARQL-query for extracting temporal dependencies from the context model.

```
SELECT DISTINCT ?actorName
WHERE {{
    ?attribute iot-lite:isActed ?actuating .
    ?attribute iot-lite:isAssociatedWith ?sensing .
    ?device ssn:hasSubsystem ?sensing .
    ?device rdfs:label '{0}' .
    ?actor ssn:hasSubsystem ?actuating .
    ?actor rdfs:label ?actorName
}}
```

Listing 4.4: SPARQL-query for extracting temporal dependencies

`actorName` returns a device which is acted on behalf of a specific sensing device. This means, a device, i.e., the actor, processes a measurement of a sensor, i.e., the sensing device. The sensing device is given by the first argument in the query. Since an actor can have more than one acting function on the same sensing device, the list is required to be distinct. The relation between the actor and the sensing device can then be expressed as a temporal dependency (Section 4.4.2).

Locality Dependencies

Listing 4.5 shows the SPARQL-query for extracting locality dependencies from the context model.

```
SELECT ?deviceName
WHERE {{
    ?device ssn:hasDeployment ?deployment .
    ?device rdfs:label ?deviceName .
    ?deployment ssn:hasLocation ?location .
    ?location rdfs:label '{0}'
}}
```

Listing 4.5: SPARQL-query for extracting locality dependencies

The query above lists every device, which is deployed at a location that is passed as an argument, to the function. For every location found in the dataset, such a query is created. Therefore, devices that are deployed on the same location can be found and locality dependencies created (see Section 4.4.3).

Monitoring Dependencies

Listing 4.6 shows the SPARQL-query for extracting monitoring dependencies from the context model.

```
SELECT ?measurementValue
WHERE {{
    ?monitor rdf:type ssn:Service .
    ?device iot-lite:exposedBy ?monitor .
    ?device rdfs:label '{0}' .
    ?monitor iot-context:hasMeasurement ?measurement .
    ?measurement iot-context:hasValue ?measurementValue .
    ?measurement iot-context:hasTimeStamp ?date .
    FILTER( ?date <= xsd:dateTime('{1}') )
}}
LIMIT 1
```

Listing 4.6: SPARQL-query for extracting monitoring dependencies

The query for monitoring dependencies (Section 4.4.3), searches for existing monitoring measurements of a given device at a given timestamp. If no result is found at that specific timestamp, the last available measurement is outputted. `measurementValue` will then contain the current latest monitoring measurement for the device. This query only succeeds if the context model is filled with live monitoring data, e.g., from the MBP.

Capability Dependencies

Listing 4.7 shows the SPARQL-query for extracting capability dependencies from the context model.

```
SELECT ?type ?value
WHERE {{
    ?sensor rdfs:label '{0}' .
    ?sensor ssn:hasMetadata ?meta .
    ?meta iot-lite:metadataType ?type .
    ?meta iot-lite:metadataValue ?value .
}}
```

Listing 4.7: SPARQL-query for extracting capability dependencies

For a given sensor, this query returns all available capabilities. This includes the `type` and the corresponding `value` of the meta data. Therefore any sensor can be related with its capabilities (see Section 4.4.3).

4.6 Step 3: Data Cleaning

The last step of the presented concept (Figure 4.1) is the data cleaning process. To include the above defined new types of OFDs in this procedure, they need to be evaluated on the dataset. The extracted OFDs are now passed to an evaluation script which processes them. This step is done in the beginning of data cleaning. For every row in the dataset, no matter if already present or coming in as live data, every parsed dependency is checked whether it is fulfilled or not.

Considering the example depicted in Figure 4.11: I assume the denial dependency, depicted in Figure 4.4, *County* → *State* and the first entries of some dataset, containing the column *County* and *State*. The definition of a denial dependency and its violation criteria can be found in Section 4.4.1. Figure 4.11 illustrates the process of the general evaluation for an OFD. To exploit the OFDs extracted from the context model, an OFD evaluator is added, which evaluates every tuple of rows in the dataset against the OFD. The condition that a specific OFD is violated, is described in the subsections of Section 4.4 individually. Specifically, in the OFD evaluation, a Boolean feature is created for each OFD dependency, stating whether the dependency is fulfilled for each row in the data. In the depicted example, firstly, the data tuple $t_1[County]$ = Lassen County, $t_1[State]$ = Nevada and $t_2[County]$ = Lassen County, $t_2[State]$ = California is considered. Since $t_1[County] = t_2[County]$ holds, and the denial dependency $County \rightarrow State$ exists, the tuple is considered erroneous, if $t_1[State] \neq t_2[State]$. This is the case, therefore, the tuple is marked as faulty. In the second iteration, the next data tuple is viewed: $t_1[County]$ = Lassen County = $t_2[County]$ and $t_1[State]$ = Nevada = $t_2[State]$ and therefore not considered as false.

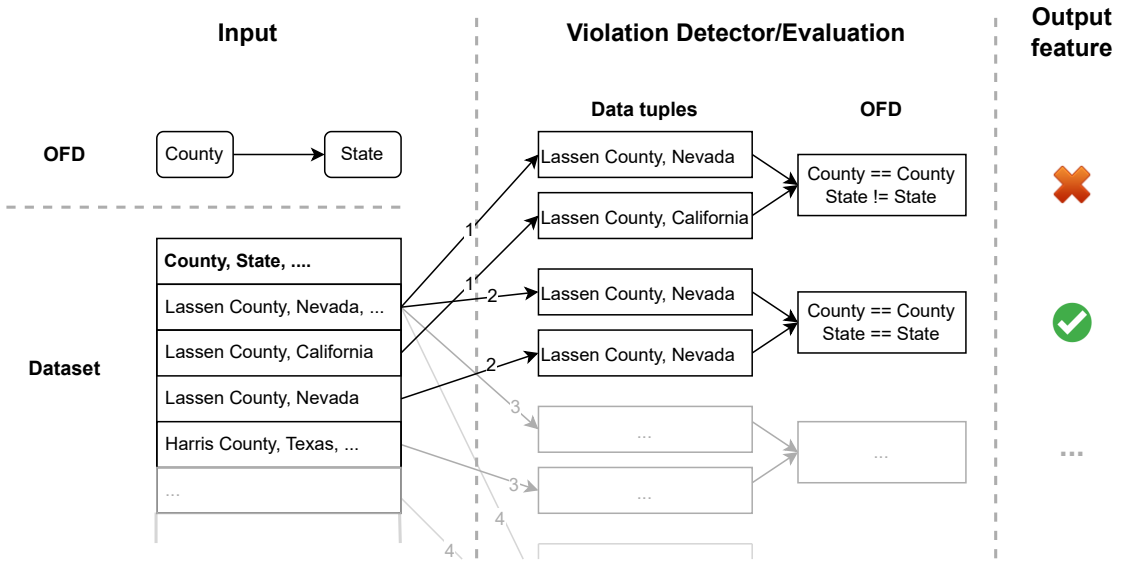

Figure 4.11: Example for OFD evaluation

In the further data cleaning pipeline, the concept presented in this thesis, relies on a state-of-the-art data cleaning framework, referred to as HoloClean [RCIR17]. However, it is important to mention that this approach can be integrated with any other data cleaning method. Structure-based OFDs,

i.e., the denial dependencies and matching dependencies, are evaluated with the existing techniques from HoloClean [RCIR17]. If so, the corresponding cells are marked as potential erroneous and further processed in the repairing pipeline of HoloClean.

Other OFDs are evaluated with the newly introduced OFD evaluation process. Since the device-link dependency, the capability dependency and the monitoring dependency do not need any related measurements or data entries, they are evaluated solely per row. If any dependency is violated, for example, a sensor has a measurement which is outside the range of its specification and therefore violates a capability dependency, the corresponding cell is marked as faulty.
For the last two dependencies, locality dependency and temporal dependency, other rows from the past need to be considered for evaluation. Due to performance reasons, it does not make sense to use every existing pair of row in the dataset. Therefore, a time-shift window is created. This window can be adjusted manually via a parameter and denotes the starting time, where relating entries in the dataset are used to evaluate the OFDs. The end of the interval is usually the timestamp of the current database entry. In this window, it is reasonable to search for measurements of nearby sensors or data points of temporal predecessors. If such an entry is found, the temporal dependency can be evaluated. For the outlier detection, using locality dependencies, any function can be used whether a measurement is an outlier or not, e.g., with a static threshold. This method can be tuned manually or can be improved to support a more complex approach to detect outliers. In Section 7.1, an idea is sketched to improve outlier detection further by measuring distances between sensors, using locality dependencies.

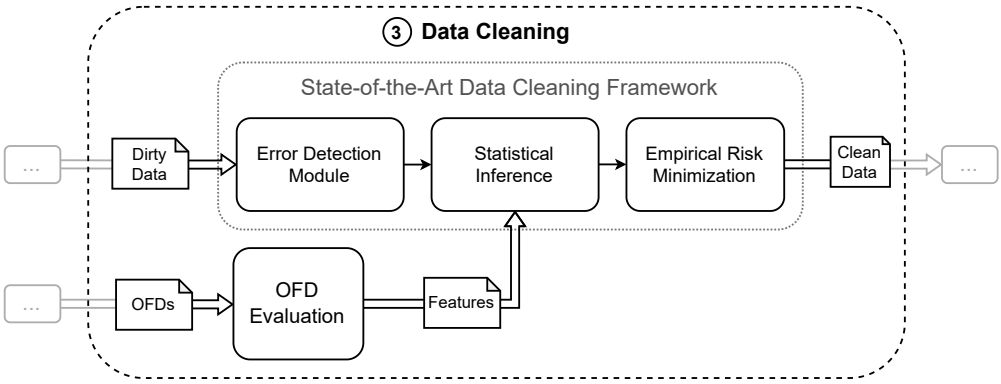

Figure 4.12: Overview of the components in Step 3: Data Cleaning

Figure 4.12 depicts the third step of the proposed pipeline, namely data cleaning. The binary features, produced by the OFD evaluation, serve as an input for a state-of-the-art data cleaning framework, e.g., HoloClean. The proposed pipeline will then use DeepDive[10], a declarative probabilistic inference framework to run statistical learning and inference. Inference rules are constructed in Differential Datalog (DDlog), a declarative language for incremental computation [RCIR17]. The output DDlog rules define a probabilistic program which is then evaluated using the DeepDive framework. Finally, the process of data repairing can be done, using statistical learning

[10] http://deepdive.stanford.edu/

and inference. Variables that correspond to clean cells are treated as labeled examples to learn the parameters of the model [RCIR17]. Using empirical risk minimization, noisy cells are inferred and therefore, the dataset repaired correspondingly.

Since the extraction and evaluation of OFDs is fully implemented in the prototype of this thesis, its output can be easily converted as an input to any arbitrary error detection framework. It can even be used stand-alone to only detect and mark errors in a dataset, given the context model. In the chapter about the implementation (Section 5.2.1), more details about HoloClean and why it is used by the presented concept, are explained.

5 Prototype and Implementation

To evaluate the concept presented in Chapter 4, a prototype has been implemented which is described in this chapter. In the beginning (Section 5.1), two datasets are introduced. Parts of them were already used by examples in previous sections. The datasets are used to evaluate the implementation about the newly proposed data validation pipeline. The goal of this prototype is to show two different use cases of the presented concepts: (a) general application and (b) IoT application. While for (a) a in the literature commonly used dataset, Hospital (Section 5.1.1), is considered, for the IoT application, data is collected from a self-deployed IoT environment (Section 5.1.3). These two different datasets are then used for evaluation and performance comparisons in the next chapter (Chapter 6). Further, in Section 5.2, a more detailed description of the used state-of-the-art data cleaning framework, HoloClean, is given. Along with this, used python packages and other tools, which are needed for the prototype or later in the evaluation, are presented (Section 5.2.2).

5.1 Datasets and Ontologies

The evaluation of the presented concepts will be done on two different datasets. The Hospital dataset, commonly used as a benchmark dataset for data cleaning algorithms [GR19; HMIR19; MAF+19; VA18], and a dataset from an IoT environment, described in Section 5.1.3. The evaluation with the hospital dataset should proof the applicability of the concepts on a generic dataset, containing mostly categorical values. Since, especially in IoT environments, datasets consists mainly of numerical values, the evaluation is done as well with an IoT dataset. Furthermore, in the latter, errors are injected using a error generator tool. In this section, these two datasets are introduced. For both datasets, ontologies are created which will then be used for the context-aware data validation done by the prototype.

5.1.1 Hospital Dataset

Hospital is a real-world medicare dataset from the US health service [DEE+13]. Typos are introduced artificially in approximately 5% of its 19,000 cells over 1000 rows and 19 columns. Each row reports the performance of a particular hospital on a specific metric. Additionally, it includes metadata such as hospital address and phone number. Since the same hospital appears multiple times, duplication of values in a column exists, which can help in the data cleaning process. Table 5.1 shows an excerpt of the dataset over only 5 from the 19 columns.

HospitalName	City	Zip Code	PhoneNumber	MeasureCode	...
helen keller memorial hospital	sheffield	35660	2563864556	ami-1	...
southeast alabama medical center	dothan	36302	3347938701	ami-1	...
marshall medical center south	boaz	x5957	2565938310	ami-1	...
eliza coffee memorial hospital	florence	35631	2567688400	ami-3	...
mizell memorial hospital	opp	36467	3344933541	ami-3	...
crenshaw community hospital	luverne	36049	3343353374	ami-3	...
st vincents east	birmingham	35235	2058383122	ami-7a	...
...	...	...	...	...	...

Table 5.1: Excerpt of the hospital dataset

5.1.2 Hospital Ontology

To represent the structure of the hospital dataset, an ontology is constructed. Therefore, the overall structure is analyzed manually. Table 5.2 depicts new classes, attributes and relations that are used for modeling the context in an ontology. A new namespace, namely `hosp` is introduced. From real-world relations between the columns, e.g., a *city* can have several *zip codes*, the model is build from the ground up.

type	name
class	hosp:Hospital
class	hosp:Address
class	hosp:ZipArea
class	hosp:City
class	hosp:County
class	hosp:State
class	hosp:EmergencyService
class	hosp:HospitalType
class	hosp:Condition
class	hosp:MeasureCode

type	name
attribute	hosp:HospitalName
attribute	hosp:PhoneNumber
attribute	hosp:HospitalOwner
attribute	hosp:ProviderNumber
attribute	hosp:Address1
attribute	hosp:ZipCode
attribute	hosp:City
attribute	hosp:EmergencyService
attribute	hosp:CountyName
attribute	hosp:State
attribute	hosp:Condition
attribute	hosp:MeasureCode
attribute	hosp:MeasureName
attribute	hosp:Stateavg

type	name
relation	hosp:hasAddress
relation	hosp:hasZipArea
relation	hosp:hasCity
relation	hosp:hasCounty
relation	hosp:hasState
relation	hosp:hasEmergencyService
relation	hosp:hasType
relation	hosp:hasMeasure

Table 5.2: Additionally used classes, attributes and relations in the hospital ontology

Figure 5.1 represents the context model for the hospital dataset. The columns of the dataset are matched with the names of the properties of the newly created classes. Therefore, the evaluation of the extracted OFDs is easy, due to no naming difference between the dataset and the ontology. The four columns, namely *HositalName*, *PhoneNumber*, *HospitalOwner*, *ProviderNumber* are attributes describing a hospital instance, thus grouped into properties of the corresponding class in the ontology. The address of an hospital is divided from more detailed to rough information in a class hierarchy. An emergency service, accordingly to the dataset, is assigned to an zip area, where as a hospital type is a refinement of the hospital class. The patients condition is classified based on a measure, which is described by a name, the measure code and the average state. With this information the presented ontology is created.

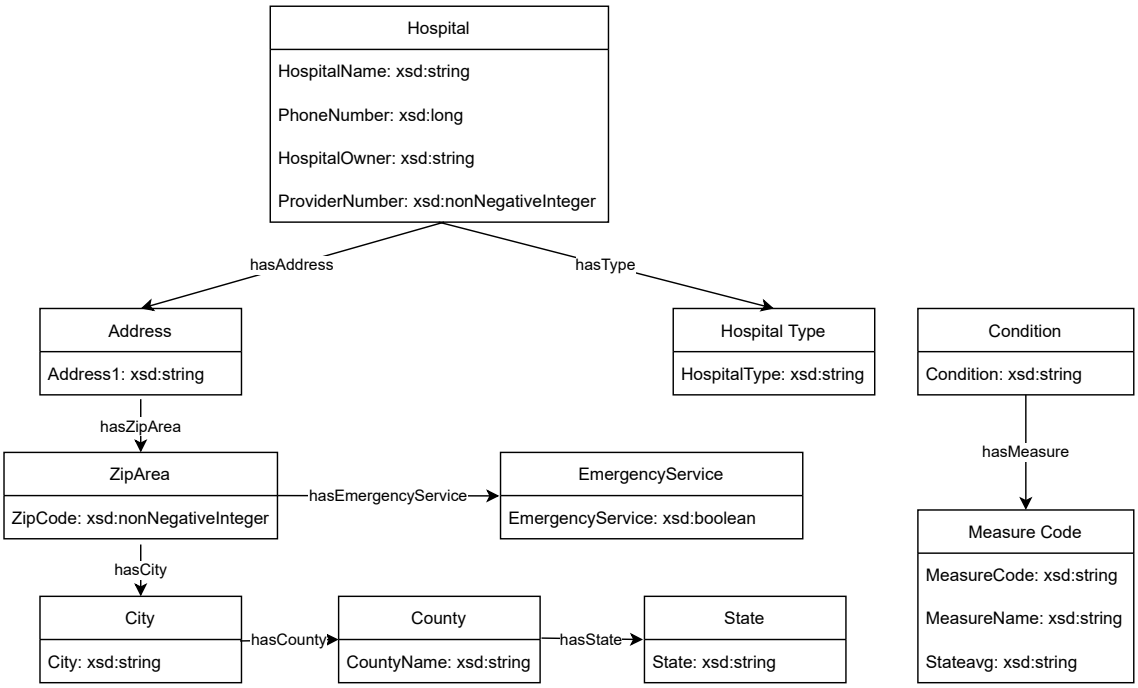

Figure 5.1: Ontology to represent structure of hospital dataset

5.1.3 IoT Dataset

For implementing and evaluating the concept of using context to detect errors in IoT environments, a real-world dataset is needed. Most datasets available online only contain the data itself but no information about the context or the environment model, the data was created in. To overcome this problem, this thesis uses a self-recorded dataset from a smart home application.

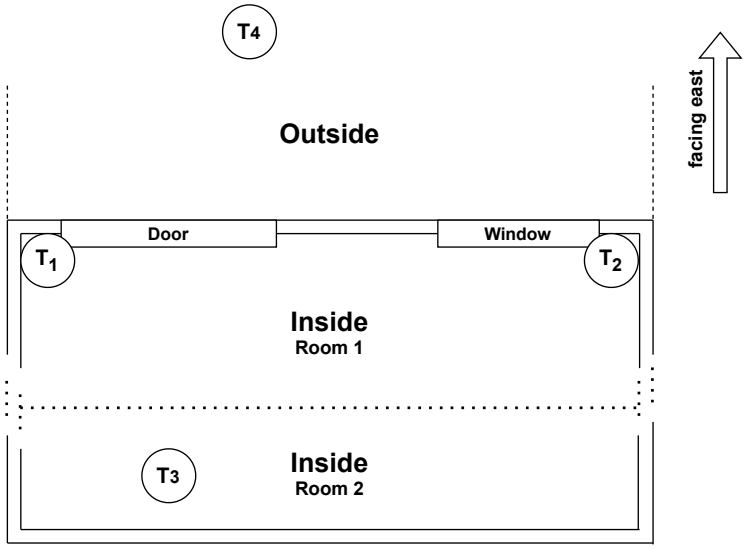

Figure 5.2: Environment model of smart home

Figure 5.2 shows an overview about the locality and distribution of the smart devices. All four sensors are temperature sensors which are mounted on the wall in the corresponding room. T_1 and T_2 are *DS18B20* 1-wire digital thermometers, produced by *maxim integrated*[1]. They are connected via 2.4 GHz WiFi[2] to a central data collecting device. Sensors T_3 and T_4, with model number $WSDCGQ11LM$ are made by $Aqara$[3]. These are wireless temperature and humidity sensors which are connected via ZigBee, an IEEE 802.15.4 standard[4]. The IoT dataset contains temperature read-outs every hour. If more than one value is recorded in this interval, the mean is calculated. The data set is mostly free from errors, but it contains *null* values and the read-out value of -128 is resulted when a sensor is faulty. These already existing errors are removed manually before the evaluation to have cleaner results for comparison (Chapter 6).

Error injection

The IoT dataset is clean from errors, thus, errors needs to be injected artificially. To inject realistic errors, a tool, *error-generator*[5], is used. The errors comprise *typos*, *value errors*, and *null values*. The injected *typos* are based on the python library *butter-fingers*[6], which generates highly realistic typos. To inject *value* errors, random values from the original value range of this column are used. They are only injected on the numerical columns of the dataset. In summary, 5% errors of each category is injected, resulting in 149 erroneous instances. Furthermore, outliers are injected into the numerical values of the IoT dataset to compare the approach with a typical outlier detection method. Two different datasets are created with numerical values and errors: (a) with random errors in the original value range, and (b) with random errors in the doubled range of the original values (see Section 6.2.2).

5.1.4 IoT Ontology

Figure 5.3 shows the IoT context model of the IoT dataset, mentioned above (Section 5.1.3). Though this model can be partly generated automatically by the MBP, if the environment is defined, the ontology is constructed manually for now. The reason for this is that the MBP does not support a fictitious environment, thus, requiring the actual hardware to be there and reachable. Moreover, the additionally introduced classes and properties in Section 4.3 are currently not supported to be extracted into an IoT context model, introduced by Del Gaudio et.al.. This feature is mentioned as future work (Section 7.1).

[1] https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf
[2] https://www.wi-fi.org/
[3] https://www.aqara.com/us/temperature_humidity_sensor.html
[4] https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf
[5] https://github.com/BigDaMa/error-generator
[6] https://github.com/alexyorke/butter-fingers

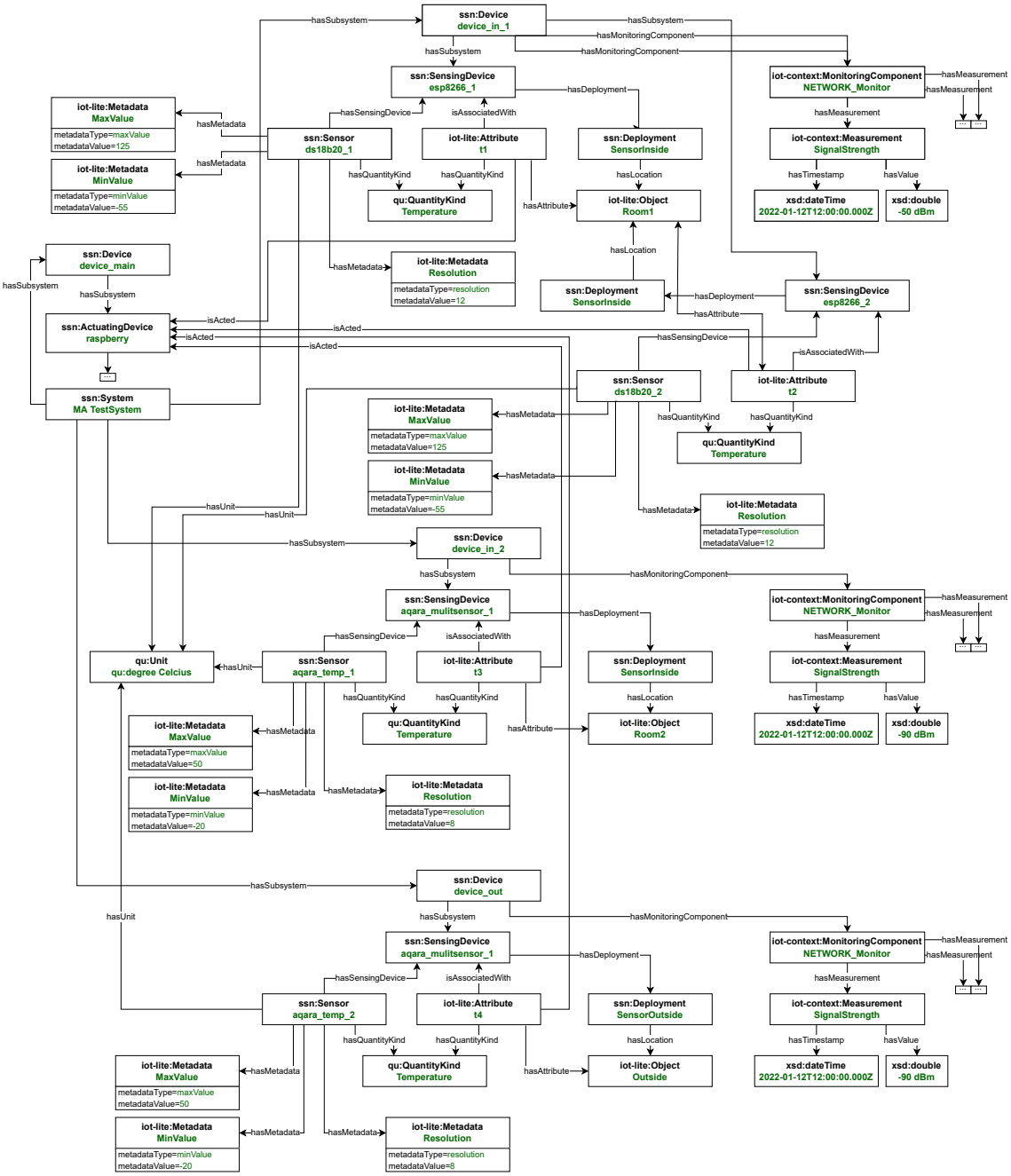

Figure 5.3: IoT context model used for the evaluation

5.2 Implementation

The implementation is built on top of the state-of-the art data cleaning framework, introduced in Section 5.2.1, referred to as HoloClean [RCIR17]. Currently, HoloClean only supports manually written constraints. This is very cumbersome due to the high effort it takes for a human to extract constraints from datasets. Especially in large datasets where every constraint can not be overlooked,

this leads to faulty or incomplete sets of constraints. My implementation extends HoloClean to discover the newly defined types of OFDs, described in Section 4.4, from context models and exploit them subsequently. In a first step, the dependencies are automatically extracted from the context model (Section 4.5). Then, the discovered dependencies are evaluated against the dataset (Section 4.6). The following describes the data cleaning framework, HoloClean, and the evaluation process of OFDs more in detail.

5.2.1 HoloClean

HoloClean [RCIR17] is a a state-of-the-art data cleaning framework by Ilyas et. al.. The workflow of HoloClean can be divided into three main parts: (a) *Error Detection*, (b) *Compilation* and (c) *Data Repairing*. These steps can also be found in Figure 5.4, where an overview is shown. A dataset to be cleaned, and manually-crafted constraints can be inputted to the framework. *Error Detection* is implemented as a black box, where the user can specify any method to detect erroneous cells. While supporting different methods for error detection, e.g., null detector and violation detector (for manually-crafted dependencies), the framework is still extensible to include new detectors implemented by the user. This functionality comes in handy when adding the OFD detector, which automatically generates and evaluates the newly defined OFDs. The proposed framework will then use DeepDive[7], a declarative probabilistic inference framework, to run statistical learning and inference in the *Compilation* step. Finally, in the *Data Repairing* step, HoloClean uses statistical learning and inference with *empirical risk minimization* to infer noisy cells and repair the dataset correspondingly [RCIR17]. The cleaned dataset is then outputted back to the user.

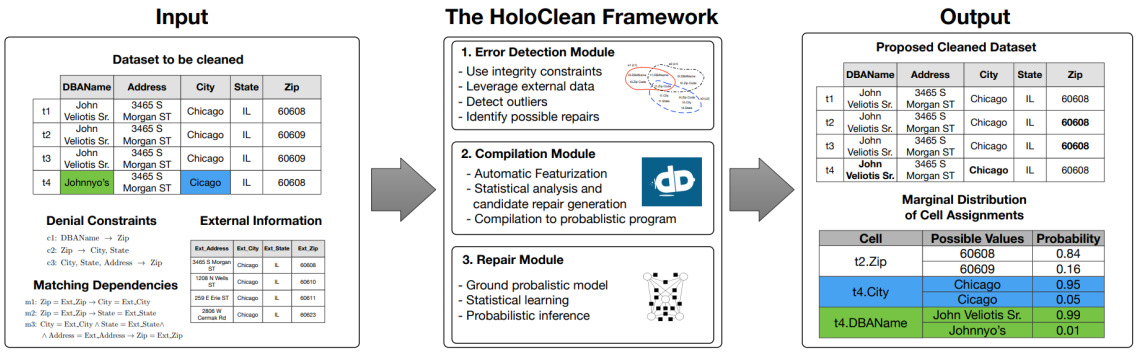

Figure 5.4: Overview of HoloClean with example dataset and a set of constraints [RCIR17]

HoloClean currently only supports denial dependencies which makes it hard to include other relations in the data for error detection methods. To be the most compatible, the implementation of this thesis uses, at least where it is possible, the file-format used by HoloClean to express constraints. This concept is applicable for *matching dependencies* (Section 4.4.1) and *denial dependencies* (Section 4.4.1) extracted from the ontology. The two structure-based OFD types are also the ones being evaluated by HoloClean itself via the *ViolationDetector*. An OFD can directly be formatted to

[7] http://deepdive.stanford.edu/

a one line string, where `EQ(X,Y)` denotes the equality operator and `IQ(X,Y)` the inequality operator. If every extracted OFD is added line by line, the file can be saved and declared as input for HoloClean.

The example for a *denial dependency* shown in Figure 4.4 can be expressed as the following, where t_1, t_2 denotes an arbitrary pair of two rows in the dataset:

```
t1&t2&EQ(t1.CountyName,t2.CountyName)&IQ(t1.State,t2.State)
```

Listing 5.1: Example for representation of a denial dependency in HoloClean

The example for the *matching dependency* depicted in Figure 4.5 needs to be expressed in two lines due to HoloClean fundamentally only supporting non symmetric dependencies. Since those are symmetric (Section 4.4.1), the OFDs needs to be generated in both directions. For simplicity only, the two first attributes are considered in this example:

```
t1&t2&EQ(t1.HospitalName,t2.HospitalName)&EQ(t1.PhoneNumber,t2.PhoneNumber)
t1&t2&EQ(t1.PhoneNumber,t2.PhoneNumber)&EQ(t1.HospitalName,t2.HospitalName)
```

Listing 5.2: Example for representation of a matching dependency in HoloClean

5.2.2 Used Tools and Libraries

In this section, tools and packages, used by the implemented prototype, are introduced and described shortly. Additionally, two error detection frameworks, Raha [MAF+19] and dBoost [PMHM16], are presented. They are used for the evaluation in Chapter 6. Figure 5.5 depicts an overview of the implementation about the used components and external libraries. The context model can either be inputted as a file or directly in a database. Several libraries are used to extract and query these ontologies. They are described in the following.

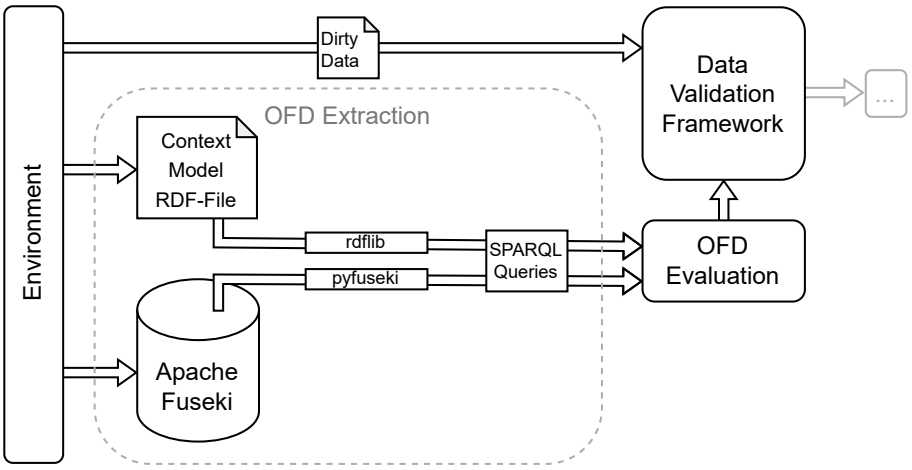

Figure 5.5: Overview of components of the prototype

Raha

Raha is a configuration-free error detection system, presented by Mahdavi et. al.[MAF+19]. The error detection framework try to solve all time-consuming steps in an automated manner, namely (a) algorithms selection, (b) algorithm configuration and (c) result verification. Different configuration sets for each error detection method are generated automatically by the framework. With a novel sampling and classification scheme for generated feature vectors, Raha chooses the most representative values for training. The detection mechanism relies among other thins on relations found in the data itself. In this thesis, Raha is used as a benchmark for modern error detection systems. Its results are compared against the presented concept of automatically extracting OFDs from context models in Section 6.2.1.

dBoost

dBoost is a tool for outlier detection in heterogeneous datasets by Pit-Claudel et. al. [PMHM16]. It uses automatic tuple expansion to characterize and locate outliers. Among other cell types, such as strings or integers, it supports for outlier detection in data series, containing floating-point numbers. The framework relies on inference and statistical modeling to flag suspicious fields in the database tuples [PMHM16]. This thesis uses dBoost for comparison to the newly presented approach, especially in outlier detection (Section 6.2.2).

SPARQL Protocol and RDF Query Language

SPARQL Protocol And RDF Query Language (SPARQL)[8] is a query language for Resource Description Framework (RDF). It supports diverse data sources in RDF format and can query graph patterns along with their conjunctions and disjunctions. W3C initiated a RDF Data Access working group which has identified use cases and requirements[9]. They define a standard way to query or access data in RDF format. RDF is a data format which represents a directed, labeled graph. It is used mainly for representing information in the web and storing ontologies. The implementation in this thesis uses SPARQL-Queries to search in ontologies for dependencies between individuals which can be used to derive OFDs from it. Those are then used for error detection and relations between them.

rdflib

rdflib[10] is a python package which can be used to parse and serialize ontologies, store implementations, and query and update an ontology. In the implementation of this thesis, *rdflib* is used to parse ontologies from a file and query them to extract information.

[8] https://www.w3.org/TR/rdf-sparql-query/
[9] https://www.w3.org/TR/rdf-dawg-uc/
[10] https://github.com/RDFLib/rdflib

Apache Jena Fuseki

Apache Jena Fuseki[11] is a SPARQL server. It can store ontologies to a database and provides the SPARQL 1.1 protocols for query and update functions. Since the IoT context model [DABS22] can automatically be generated and stored into a Fuseki database, the prototype is able to extract this information directly from there.

pyfuseki

pyfuseki[12] can be used in python to connect to a Fuseki database. It is a python package with which one can easily query stored ontologies. The package is used by the implementation to establish a connection to a Fuseki database.

[11] https://jena.apache.org/documentation/fuseki2/
[12] https://github.com/yubinCloud/pyfuseki

6 Evaluation

In this chapter, the concept of context-aware data validation, described in this thesis, is evaluated. The evaluation is divided in two parts: (a) evaluation with the Hospital dataset (Section 6.1), and (b) evaluation in an *IoT application* with the Smart Home dataset (Section 6.2). To be able to compare every run, several metrics are being evaluated. These metrics are described in the following:

Evaluation metrics

Detected errors

The total amount of detected errors in the dataset.

Total repairs

The total amount of repairs performed to the dataset, no matter if correct or false.

Correct repairs

The total amount of cells, which were false and have been repaired to the correct value.

Total repairs on incorrect cells

The total repairs performed on cells which were faulty.

Total repairs on correct cells

The total repairs performed on cells which were correct and did not need to be repaired.

Precision

The fraction of correct repairs over the total number of repairs done to the dataset.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

see Table 6.1 for explanation of acronyms

Recall

The fraction of correct repairs over the total number of errors in the dataset.

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

see Table 6.1 for explanation of acronyms

Repairing Recall

The fraction of correct repairs over the total number correctly erroneous cells identified by error detection [RCIR17].

F_1-score

The harmonic mean of precision and recall calculated as follows:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FP} + \text{FN}}$$

see Table 6.1 for explanation of acronyms

Repairing F_1-score

The harmonic mean of precision and repairing recall calculated analogue to the normal F_1-score [RCIR17], but using the repairing recall.

		Detected Error	
		positive	negative
Ground Truth	positive	True Positive (TP)	False Negative (FN)
	negative	False Positive (FP)	True Negative (TN)

Table 6.1: Confusion matrix

6.1 Hospital Dataset

To evaluate the involvement of the structure-based dependencies in datasets, represented in ontologies, the hospital dataset, introduced in Section 5.1.1 is used. The dataset contains 1000 rows and 19 columns as depicted in Table 6.2.
For illustration purposes, Table 6.2, shows the total number of the OFDs that are automatically extracted from the Hospital dataset and some examples, categorized by their types using the concepts presented in this thesis. Manual dependencies were specifically and manually created by Rekatsinas et. al. in their work [RCIR17].

		Hospital dataset
number of	total entries	(1000, 19)
	manual dependencies	15
	auto-extracted OFDs	25
OFDs	Denial dependencies	PhoneNumber → Address1, ZipCode → City, . . .
	Matching dependencies	ProviderNumber → PhoneNumber, Stateavg → MeasureCode, . . .

Table 6.2: Examples of the extracted OFDs from the hospital dataset

6.1.1 Evaluation with HoloClean

For comparison, HoloClean is executed in three different ways: (a) *without any dependencies* (using only the null detector that already exists in HoloClean, (b) *with manually-crafted dependencies* from [RCIR17] and (c) *with automatically extracted OFDs*, as described in Chapter 4. To emphasize the importance of a good and complete set of dependencies, another evaluation run is done with only 50% of the manually-crafted dependencies, concluding to four evaluations in total.

Table 6.3 and Figure 6.1 show the total amount of detected errors and repairs done by the data cleaning pipeline. Furthermore, additional information about the amount of correct and false repairs are visualized.

	detected errors	total repairs	correct repairs	false repairs
automatically extracted OFDs	431	322	304	18
manually-crafted dependencies (100%)	435	232	232	0
manually-crafted dependencies (50%)	213	62	62	0
no dependencies	36	36	36	0

Table 6.3: Total metrics of evaluation runs with HoloClean and the Hospital dataset

In Figure 6.1, one can see that the implementation of this thesis and the manually-crafted dependencies detect a nearly equally amount of errors in the dataset. If the evaluation is done with only half or even no dependencies, the number of detected errors decrease significantly. While the same patterns apply to the repairs done on the dataset, one can see that the run with the automatically extracted OFDs compares better to the manually-crafted ones. An increase from 232 to 304 correctly repaired cells can be measured.

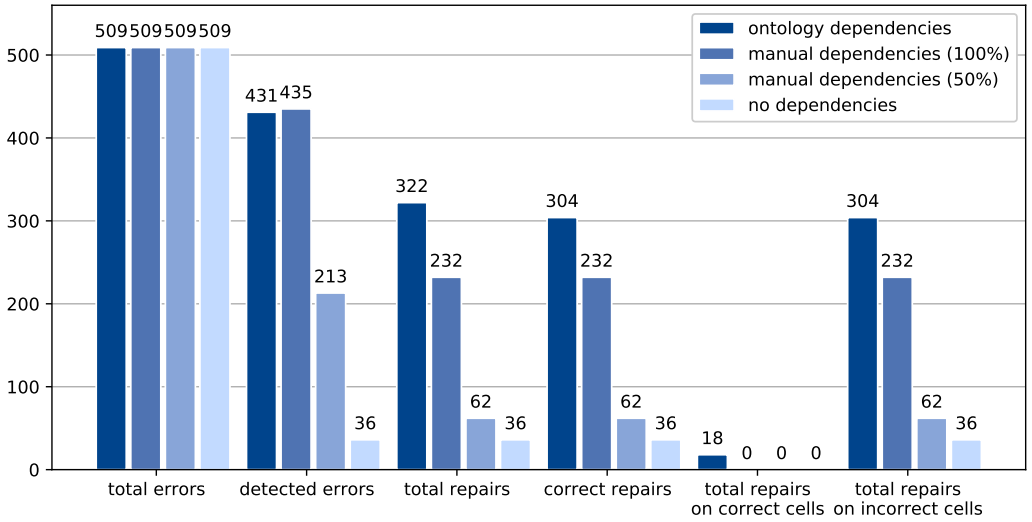

Figure 6.1: Total metrics of HoloClean evaluation with the Hospital dataset

The evaluation metrics, introduced at the beginning of the chapter, comprise the detection precision, recall and F1-score. Additionally, the repair recall and the repair F_1-score is employed. Table 6.4 and Figure 6.2 depict the evaluated metrics for the individual evaluation.

	precision	recall	F_1	repairing recall	repairing F_1
automatically extracted OFDs	0.94	0.6	0.73	0.71	0.81
manually-crafted dependencies (100%)	1.0	0.46	0.63	0.53	0.7
manually-crafted dependencies (50%)	1.0	0.12	0.22	0.29	0.45
no dependencies	1.0	0.07	0.13	1.0	1.0

Table 6.4: Fraction metrics of evaluation runs with HoloClean and the Hospital dataset

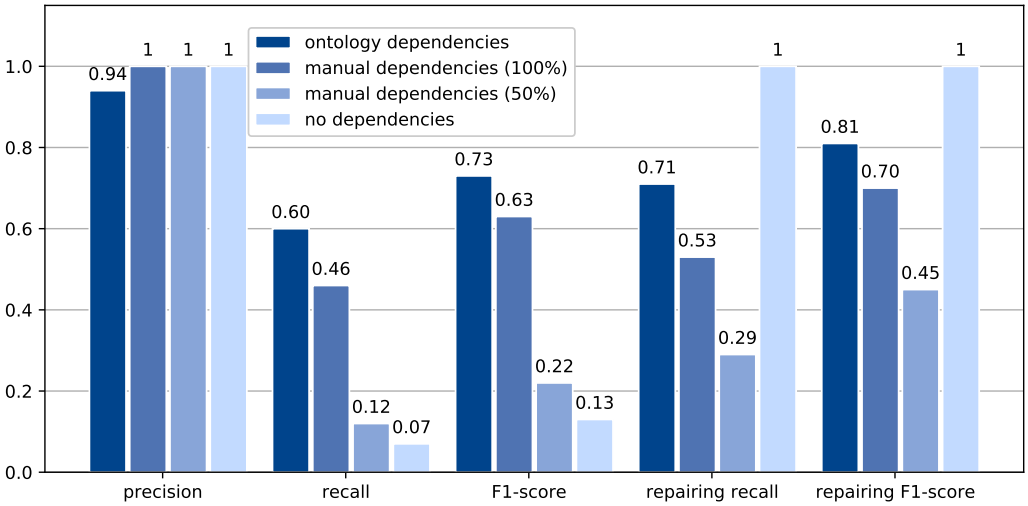

Figure 6.2: Fraction metrics of HoloClean evaluation with the Hospital dataset

Figure 6.2 depicts some minor differences, i.e., about 6%, in the precision of detected errors throughout the different runs, but shows a significant decrease in recall and therefore also in the F_1-score. Considering less or even no dependencies for data validation is much worse and leads to marking only about 7% of the faulty cells as errors. Since the run with the manually-crafted dependencies performs only a little worse than the automatically generated ones, one can say, that those dependencies describe the relations in the dataset, at least with the same quality than the manual dependencies. Nonetheless, the concept, presented in this thesis, which is depicted as the "ontology dependencies" data series, has a higher performance in recall and F_1-score than any other method.

One can determine the high repairing recall and, thus, the resulting high F_1-score in the evaluation with no dependencies. This is caused by the high precision and low total number of detected errors when running HoloClean without dependencies. Since the dependencies have been created specifically for the errors in the dataset, HoloClean achieves high detection precision in the case of using the manually-crafted dependencies. However, they achieve low detection recall, since the available dependencies are not sufficient to describe the important relationships in the dataset. Conversely, the implementation of this thesis yields a significant improvement in recall, F1-score, repair recall, and repair F_1-score thanks to the automatically-generated OFDs. For instance, the implemented concept achieves higher detection F_1-score, at least by 14%, relative to HoloClean with 100% dependencies.

6.2 IoT Dataset

To evaluate the presented concept of data cleaning in IoT environments, the IoT dataset, introduced in Section 5.1.3 is used. The dataset contains 1000 rows and 7 columns.
For illustration purposes, Table 6.5 shows the total number of the OFDs that are automatically extracted from the IoT dataset and some examples, categorized by their types using the concepts presented in this thesis. In contrast to the static structural relations between the columns in the Hospital dataset, the IoT dataset contains messages and measurements that are related to each other. Therefore, in this evaluation time-based and value-based dependencies can be extracted from the context model as well.

		IoT dataset
number of	total entries	(1000, 7)
	auto-extracted OFDs	21
OFDs	Denial dependencies	System → Device, Device → SensingDevice, . . .
	Device-Link dependencies	ds18b20_1 → device_in_1, . . .
	Capability dependencies	ds18b20_1 → MaxValue, ds18b20_1 → MinValue, . . .
	Locality dependencies	ds18b20_1 → Room1, ds18b20_2 → Room2, . . .
	Temporal dependencies	device_in_1 → device_main, . . .

Table 6.5: Examples of the extracted OFDs from the hospital dataset

6.2.1 Evaluation with HoloClean and Raha

To evaluate the presented concepts, HoloClean is executed in two different ways: (a) *without any dependencies* (using only the null detector already exists in HoloClean) and (b) *with automatically extracted OFDs*. Since the previous evaluation already yields, that HoloClean does not perform well without any constraints, another data validation framework, namely Raha [MAF+19] is used in an additional evaluation run. Details about Raha can be found in the chapter about the implementation and used tools in Section 5.2.2.
In contrast to the evaluation with the Hospital dataset, there are no manually-crafted dependencies in the evaluation with the IoT dataset. This is due to the manual dependencies being part of the research about HoloClean [RCIR17] and specifically created for the Hospital dataset by Rekatsinas et. al. Therefore these dependencies are only used to compare against the automatically extracted OFDs from this thesis about the Hospital dataset.

	detected errors	total repairs	correct repairs	false repairs
automatically extracted constraints	107	74	68	0
no constraints	50	50	44	0
Raha	41	-	-	-

Table 6.6: Total metrics of HoloClean and Raha evaluation with IoT dataset

Table 6.6 and Figure 6.3 show the total amount of detected errors and repairs done by the data cleaning pipeline. Furthermore, additional information about the amount of correct and false repairs are visualized. Since Raha only detects errors and does not repair them, the repairing performance does not get evaluated.

Figure 6.3 depicts that the approach presented in this thesis outperforms the other error detection methods. Neither Raha or HoloClean without any dependencies are performing well on the IoT dataset. The implemented concept doubles the amount of detected errors from 50 (41 with Raha) to 107, compared to the other methods. Also, in repairing capabilities, it performs better than HoloClean without the use of manually-crafted dependencies.

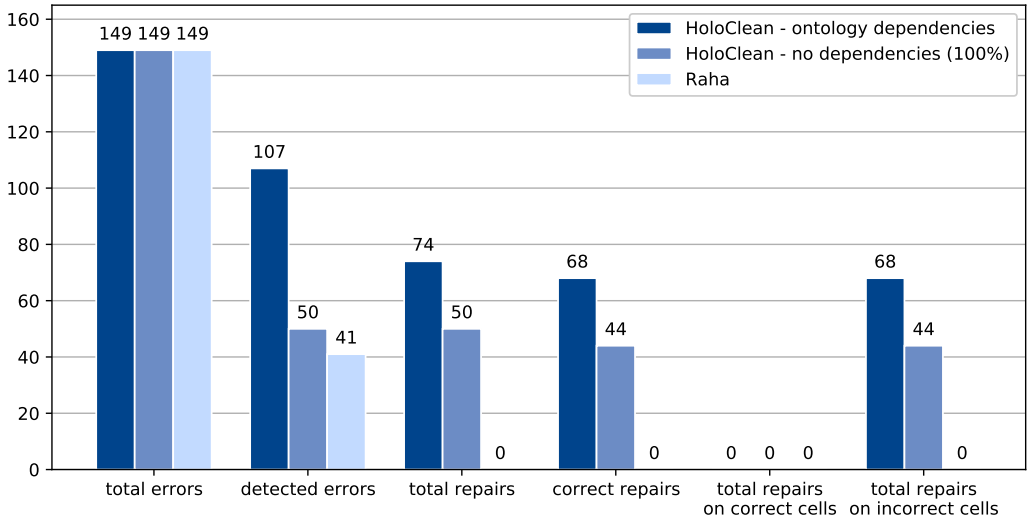

Figure 6.3: Total metrics of HoloClean and Raha evaluation with IoT dataset

The evaluation metrics, introduced at the beginning of the chapter, comprise the detection precision, recall and F1-score. The repair recall and the repair F_1-score is not evaluated on the IoT dataset, since the framework used for comparison, namely Raha, does not support repairing incorrect data cells. Table 6.7 and Figure 6.4 depict the evaluated metrics for the individual evaluation.

	precision	recall	F_1	repairing recall	repairing F_1
automatically extracted constraints	0.92	0.46	0.61	0.64	0.75
no constraints	0.88	0.3	0.44	0.88	0.88
Raha	0.72	0.28	0.4	-	-

Table 6.7: Fraction metrics of evaluation runs with HoloClean and IoT dataset

In Figure 6.4, one can see a continuously decrease in precision, recall and F_1-score, when traversing the evaluation results from "automatically extracted OFD", over "HoloClean without dependencies", to the evaluation done with Raha. The concept, presented in this thesis, has higher performance in all three metrics, precision, recall and F_1-score. While the improvement in precision is only a minor step, the performance in recall, and therefore in the F_1-score, is relatively much higher and stands out from the rest.

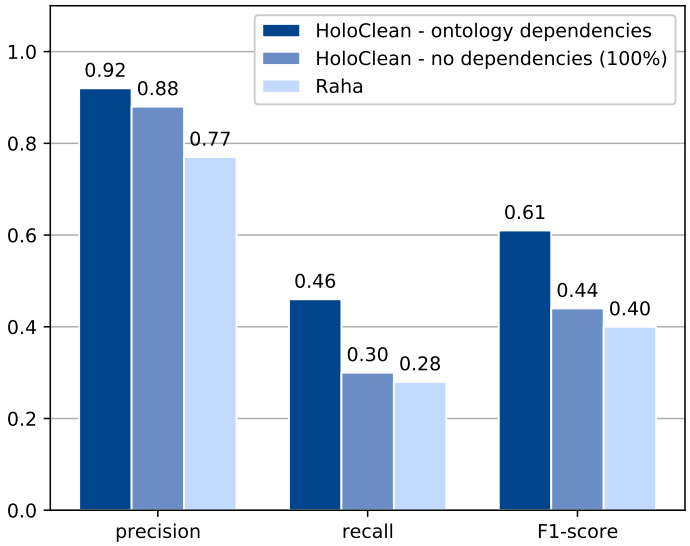

Figure 6.4: Fraction metrics of HoloClean and Raha evaluation with IoT dataset

6.2.2 Evaluation of Outlier Detection with dBoost

To measure the ability of the implemented concept to detect numerical outliers with different ranges, a comparison with dBoost[1] is done. dBoost is a tool specifically made to detect outliers (Section 5.2.2). Each method is executed with two IoT datasets, which differ in the values of the injected errors. The dataset is also collected from the IoT environment, described in Section 5.1.3, and only includes measurements from the sensor T_1.

For the run "random", the dataset is injected with errors only in the original range of values. Whereas, the label "+100%" indicates the dataset with injected errors in the doubled range of the original values. To concretize, the original dataset contains temperature values between 18.69 °C and 31.67 °C, which is exactly the range used for errors in the "random" dataset. The "+100% range" dataset is injected with errors in the range from 9.35 °C to 47.51 °C. This results in the evaluation being run with two different difficulties: (a) an easy dataset, where outliers are easier to identify, because they can lie outside the original range, and (b) a more difficult dataset, where errors are only taken from the original value range. (a) contains 83 errors in a total number of 8580 data points and (b) is injected with 84 errors with the same total number of values. Table 6.8 and

	TP	FP	TN	FN
+100% range - ontology dependencies	68	31	8465	16
+100% range - dBoost	49	0	8496	35
random - ontology dependencies	29	32	8465	54
random - dBoost	0	4	8493	83

Table 6.8: Total metrics of HoloClean and dBoost evaluation with IoT outlier dataset

[1] https://github.com/cpitclaudel/dBoost

Figure 6.5 show the total values of TP, FP, TN, and FN for the different evaluation runs. While for TP a high values is desirable, for FP and FN a low value corresponds to a better performance. My approach performs better compared to dBoost in the detecting faulty cells as an error, but does not perform well in terms of the amount of detected errors, which were actually correct. This statement reflects in the computed precision of both methods, where dBoost performs better. Nonetheless, the concept, I implemented, produces less FNs, thus performs better in recall.

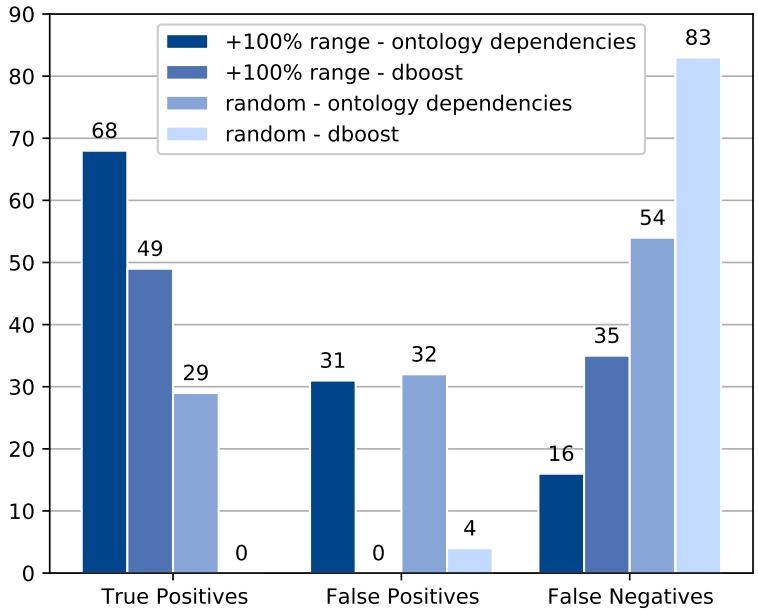

Figure 6.5: Total metrics of outlier evaluation with dBoost and IoT dataset

The evaluation metrics, introduced at the beginning of the chapter, comprise the detection precision, recall and F1-score. The repair recall and the repair F_1-score is not evaluated on the IoT outlier dataset, since dBoost, does not support repairing incorrect data cells.

Table 6.9 and Figure 6.6 depict the evaluated metrics for the individual evaluation.

	precision	recall	F_1-score
+100% range - ontology dependencies	0.69	0.81	0.74
+100% range - dBoost	1.0	0.58	0.73
random - ontology dependencies	0.48	0.35	0.4
random - dBoost	0.0	0.0	0.0

Table 6.9: Fraction metrics of HoloClean and dBoost evaluation with IoT outlier dataset

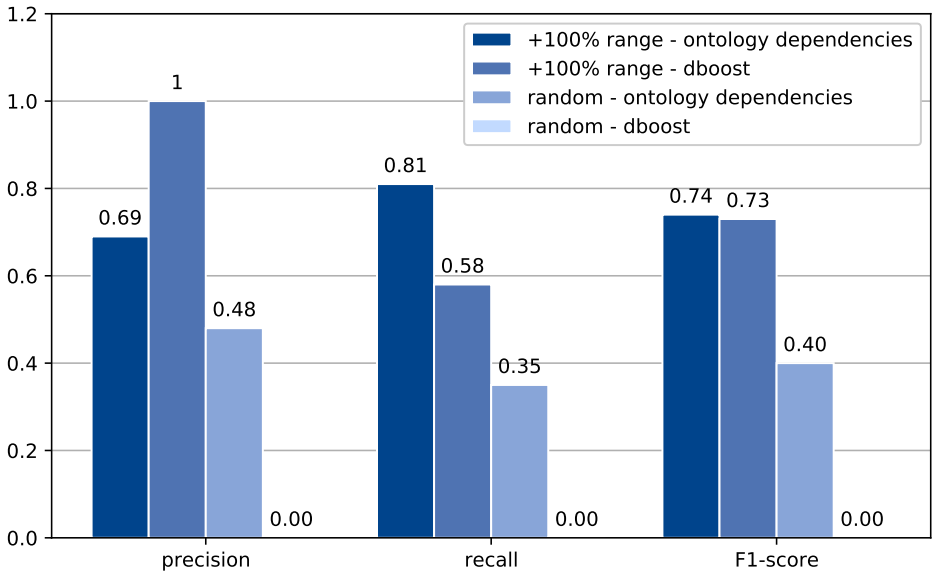

Figure 6.6: Fraction metrics of outlier evaluation with dBoost and IoT dataset

As the Figure 6.6 and Table 6.9 depict, dBoost performs well in terms of the detection precision on the "+100%"-data set. However, it fails to detect erroneous instances when they are in the original range. Conversely, the approach, presented in this thesis, can effectively detect errors in the different ranges and reaches a F_1-score of 0.4. For instance, the implemented concept achieves higher detection F_1-score (at least by 1.3% and 40%) than dBoost in the "+100%" range and the original range, respectively. On the evaluated dataset with injected errors from the doubled range, both methods perform about the same for the F_1-score. While dBoost has a higher precision on error detection, the recall is better when using the method for automatically extracting OFDs that is presented in this thesis.

6.3 Discussion of Results

The evaluation shows that the proposed concept and pipeline for the data validation process performs better than typical state-of-the-art error detection methods, i.e., increasing the F_1-score in the evaluation with the IoT dataset by 0.17 to 0.61. Moreover, the evaluation in outlier detection shows that errors inside the typical range of the data can be detected in cases, where other techniques fail. dBoost does not detect any true error in the "random" evaluation, while the presented concept still achieves a recall of 35%. The concept is capable to dynamically react to changes in the environment where the data originates from, e.g., a change in the topology or a sensor failure. However, it requires the context model to always be up-to-date and to represent the current state of the environment at all time. This process currently still involves some manual changes, due to the automated context

model generation being not fully implemented yet. Nonetheless, the extension to the work of Del Gaudio et. al. [DABS22] can be automated in future work to have a fully-automated context-aware data validation pipeline.

As mentioned in Section 6.1.1, the detection and repair accuracies are improved when increasing the number of manual dependencies from 0% to 50% and from 50% to 100%. The total number decreases from 435 by half to only 213 total detected errors. Such a result emphasizes the general impact of using dependencies for data cleaning. As mentioned, the high precision of HoloClean is caused by the manually-crafted dependencies being created specifically for the errors in the dataset. Therefore, it can be said that well-tuned dependencies can have a big positive impact on the results of the data cleaning pipeline. For example, the F_1 score nearly doubles from 0.22 to 0.63 when evaluating HoloClean with twice as many dependencies. By extracting OFDs automatically from a model of the environment where the data originates from, it is ensured that the dependencies have a good quality and therefore error detection performs better.

Although the errors are injected artificially in the datasets, they were added using a commonly used tool based on highly realistic typos and values from the same domain. Therefore, it can be said that the injected errors are similar to errors that would occur in the real world. Nonetheless, an evaluation with a real world dataset containing several errors is considered as future work.

The IoT dataset, used for the evaluation in Section 6.2, does not contain any monitoring dependencies. Monitoring dependencies can be used in various of ways to enhance error detection. For example in wireless networks, the signal quality can be a valid indicator for the probability of a false transmitted value. Additionally, the health of the participating devices and sensors can be monitored and used to develop algorithms which output a confidence that a certain value is measured by a faulty device and therefore erroneous. Due to the fact that almost every IoT dataset lacks of monitoring information and the time limit of this thesis, monitoring dependencies are not evaluated. The assumption is made that if those were considered, the performance of the presented prototype will not be degraded.

Nonetheless, it must be said that the presented approach only performs well if OFDs, e.g., for the outlier detection especially the *locality dependency*, can be discovered in the environment. For the evaluation with the Hospital dataset, 25 dependencies could be discovered, while there have been discovered 21 OFDs in the IoT dataset. If there is no dependency that can be extracted and used for the error detection process, the method can not improve error detection from already existing techniques. This means that there should be at least any context in the environment, i.e., a network of data generating devices, instead of only one device. Nonetheless, in the modern world, technologies in various application fields are equipped with an increasing number of smart devices, e.g., sensors for redundancy [BA18; MR17; YWS17], which leads to a higher chance of discovering more dependencies in the environment.

7 Summary and Future Work

In this thesis, a concept for automated tabular data cleaning powered by dynamic functional dependency rules extracted from a context model has been introduced. To achieve this, a data validation pipeline is proposed which consists of the three steps: context modeling, OFD extraction, and data cleaning. An ontology-based context model has been used which carries knowledge about the environment the data originates from and therefore provides dependencies and constraints about it. The dependencies are extracted automatically and evaluated on the dataset to improve current approaches in data validation.

As motivated in the Smart Home scenario in Section 1.1 and in the introduction, nowadays, there is a need for data with the least amount of erroneous data entries as possible. Machine learning algorithms require lots of training data to build a model and use this for, e.g., prediction tasks. The performance of the prediction algorithm is directly caused by the quality of the input data. Therefore, there is a high interest in good data cleaning methods. As mentioned, this thesis has shown, how existing methods can be enhanced and improved, when considering the context in form of a context model about the environment.

The context model is based on an ontology, extended with classes and properties from other commonly used namespaces to express dependencies. Live updates in the environment are reflected using adapters or monitoring services of modern IoT platform tools. They are adding or modifying content and relations in the context model to ensure it always represents the current state of the environment.

Using the live context model, the proposed concept can automatically generate multiple OFDs. Therefore, it broadly relieves the burden of creating reliable dependencies describing the relationships in the data. New types of OFDs are defined, which not only describe structural relations in the dataset (*denial dependency* and *matching dependency*), but also covers time-based relations (*temporal dependency*), as well as, relations between the actual value of measurements in IoT environments (*locality dependency*, *monitoring dependency*, and *capability dependency*). These OFDs are extracted from the context model with SPARQL queries and evaluated on the dataset.

In the process of data cleaning, the OFD evaluation outputs Boolean features that represent the validity of a certain cell. Modern state-of-the-art data cleaning frameworks, e.g., HoloClean, can use those features for statistical learning inference. The proposed pipeline uses such a framework to automatically analyze and repair the detected errors with probabilistic inference and empirical risk minimization.

The evaluation shows that the structural-based OFDs can be applied to various types of datasets. Since this thesis focuses on IoT datasets, the other newly introduced types of OFDs are primarily applicable in datasets from the IoT. In comparison to other typical state-of-the-art error detection methods, the proposed pipeline for the data validation process performs better. Moreover, the

evaluation in outlier detection shows that errors inside the value range of the data can be detected in cases, where other techniques fail. With the newly introduced types of OFDs, an improvement of about 15% can be achieved in the IoT dataset from a smart home application.

7.1 Future Work

In future work, it is planned to evaluate the proposed data validation pipeline end-to-end. In the evaluation of this thesis, only the performance of cleaning erroneous entries from datasets are measured. Nonetheless, it is important to know how the improved data cleaning method impacts on the downstream applications, e.g., machine learning algorithms. As depicted in the motivation scenario (Section 1.1), prediction applications have a need for data of good quality to learn and build their models from. Its prediction performance is highly related to the quality of data which is inputted. Such an end-to-end machine learning pipeline can be evaluated in future work.

The error detection mechanism can further be improved by considering other types of features when evaluating the proposed OFDs. For example, the locality dependency could be further improved if considering not only the placement of the sensor but also the actual distance between them as a feature. Therefore, the outlier detection can be improved, but it also requires the context model and the automated generation of such to be extended. Moreover, the IoT platform tools need to implement such measure.

In current research and tools, e.g. Raha [MAF+19], features for error detection are automatically gathered and learned from the dataset. These features try to represent the structure of the data. In possible future work, this approach can be combined with the proposed dependency extraction from context models to validate and/or improve dependencies. Therefore, the quality of the dependencies can be further improved, which might lead to higher performance in the overall error detection task.

It is planned that this thesis will be published as a scientific paper.

Acknowledgment

I would like to thank Prof. Dr. rer. nat. habil. Holger Schwarz for the opportunity to do my master thesis at the Institute for Parallel and Distributed Systems.

I'm extremely grateful to Daniel Del Gaudio for the friendly supervision and the motivating words during the preparation of my master thesis. Thank you for the proofreading and the valuable suggestions in shaping the thesis.

Many thanks to Mohamed Abdelaal from the Software AG, who supported me with valuable ideas and current state-of-the-art tools in the process of the evaluation of my concept.

Bibliography

[AZC+18] M. Alipour-Langouri, Z. Zheng, F. Chiang, L. Golab, J. Szlichta. "Contextual Data Cleaning". In: *2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW)*. 2018, pp. 21–24. DOI: 10.1109/ICDEW.2018.00010 (cit. on pp. 25, 26, 28).

[BA18] J. Bauer, N. Aschenbruck. "Design and implementation of an agricultural monitoring system for smart farming". In: *2018 IoT Vertical and Topical Summit on Agriculture-Tuscany (IOT Tuscany)*. IEEE. 2018, pp. 1–6 (cit. on p. 70).

[BCC20] M. Bansal, I. Chana, S. Clarke. "A Survey on IoT Big Data: Current Status, 13 V's Challenges, and Future Directions". In: *ACM Comput. Surv.* 53.6 (Dec. 2020). ISSN: 0360-0300. DOI: 10.1145/3419634. URL: https://doi.org/10.1145/3419634 (cit. on pp. 15, 22, 23).

[BEBT15] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, K. Taylor. "IoT-Lite Ontology". In: *W3C member submission* 26 (Nov. 2015) (cit. on p. 36).

[BEBT16] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, K. Taylor. "IoT-Lite: A Lightweight Semantic Model for the Internet of Things". In: July 2016. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0035 (cit. on pp. 29, 36).

[Bec16] T. Becker. "Big Data Usage". In: *New Horizons for a Data-Driven Economy: A Roadmap for Usage and Exploitation of Big Data in Europe*. Ed. by J. M. Cavanillas, E. Curry, W. Wahlster. Cham: Springer International Publishing, 2016, pp. 143–165. ISBN: 978-3-319-21569-3. DOI: 10.1007/978-3-319-21569-3_8. URL: https://doi.org/10.1007/978-3-319-21569-3_8 (cit. on p. 15).

[BKC+17] S. Baskaran, A. Keller, F. Chiang, L. Golab, J. Szlichta. "Efficient Discovery of Ontology Functional Dependencies". In: Nov. 2017, pp. 1847–1856. DOI: 10.1145/3132847.3132879 (cit. on pp. 25, 28, 31).

[BV81] C. Beeri, M. Y. Vardi. "The implication problem for data dependencies". In: *Automata, Languages and Programming*. Ed. by S. Even, O. Kariv. Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, pp. 73–85 (cit. on p. 24).

[BV98] S. Biaz, N. Vaidya. "Distinguishing congestion losses from wireless transmission losses: a negative result". In: *Proceedings 7th International Conference on Computer Communications and Networks (Cat. No.98EX226)*. 1998, pp. 722–731. DOI: 10.1109/ICCCN.1998.998834 (cit. on p. 43).

[CBB+12] M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. Le Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, K. Taylor. "The SSN ontology of the W3C semantic sensor network incubator group". In: *Journal of Web Semantics* 17 (2012), pp. 25–32. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2012.05.003. URL: https://www.sciencedirect.com/science/article/pii/S1570826812000571 (cit. on pp. 29, 35).

[CFG+07] G. Cong, W. Fan, F. Geerts, X. Jia, S. Ma. "Improving Data Quality: Consistency and Accuracy." In: Jan. 2007, pp. 315–326 (cit. on pp. 25, 27, 31).

[CIKW16] X. Chu, I. F. Ilyas, S. Krishnan, J. Wang. "Data Cleaning: Overview and Emerging Challenges". In: SIGMOD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 2201–2206. ISBN: 9781450335317. DOI: 10.1145/2882903.2912574. URL: https://doi.org/10.1145/2882903.2912574 (cit. on pp. 16, 27, 32).

[CNC+18] T. Chakraborty, A. Nambi, R. Chandra, R. Sharma, M. Swaminathan, Z. Kapetanovic, J. Appavoo. "Fall-curve: A novel primitive for IoT Fault Detection and Isolation". In: Nov. 2018, pp. 95–107. DOI: 10.1145/3274783.3274853 (cit. on p. 23).

[CPIR14] J. R. Cardoso, L. M. Pereira, M. D. Iversen, A. L. Ramos. "What is gold standard and what is ground truth?" In: *Dental press journal of orthodontics* 19 (2014), pp. 27–30 (cit. on p. 23).

[DABS22] D. Del Gaudio, B. Ariguib, A. Bartenbach, G. Solakis. "A live context model for semantic reasoning in IoT applications". In: *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 2022, pp. 322–327. DOI: 10.1109/PerComWorkshops53856.2022.9767267 (cit. on pp. 17, 18, 21, 24, 29, 31–34, 43, 59, 70).

[DEE+13] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, N. Tang. "NADEEF: A Commodity Data Cleaning System". In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD '13. New York, New York, USA: Association for Computing Machinery, 2013, pp. 541–552. ISBN: 9781450320375. DOI: 10.1145/2463676.2465327. URL: https://doi.org/10.1145/2463676.2465327 (cit. on pp. 16, 51).

[DP11] W. Dargie, C. Poellabauer. *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Jan. 2011. DOI: 10.1002/9780470666388 (cit. on p. 21).

[FHM19] A. C. Franco da Silva, P. Hirmer, B. Mitschang. "Model-Based Operator Placement for Data Processing in IoT Environments". In: *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. 2019, pp. 439–443 (cit. on pp. 29, 32–34).

[FHPM18] A. C. Franco da Silva, P. Hirmer, R. K. Peres, B. Mitschang. "An Approach for CEP Query Shipping to Support Distributed IoT Environments". In: *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 2018, pp. 247–252 (cit. on pp. 29, 32–34).

[FP05] E. Ferro, F. Potorti. "Bluetooth and Wi-Fi wireless protocols: a survey and a comparison". In: *IEEE Wireless Communications* 12.1 (2005), pp. 12–26 (cit. on p. 23).

[GR19] Z. Guo, T. Rekatsinas. "Learning functional dependencies with sparse regression". In: *arXiv preprint arXiv:1905.01425* (2019) (cit. on p. 51).

[Gru93] T. R. Gruber. "A translation approach to portable ontology specifications". In: *Knowledge Acquisition* 5.2 (1993), pp. 199–220. ISSN: 1042-8143. DOI: https://doi.org/10.1006/knac.1993.1008. URL: https://www.sciencedirect.com/science/article/pii/S1042814383710083 (cit. on p. 24).

[Haw80] D. M. Hawkins. *Identification of outliers*. Vol. 11. Springer, 1980 (cit. on p. 24).

[HMIR19] A. Heidari, J. McGrath, I. F. Ilyas, T. Rekatsinas. "HoloDetect: Few-Shot Learning for Error Detection". In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 829–846. ISBN: 9781450356435. DOI: 10.1145/3299869.3319888. URL: https://doi.org/10.1145/3299869.3319888 (cit. on p. 51).

[HZA+06] M. Hefke, V. Zacharias, A. Abecker, Q. Wang, E. Biesalski, M. Breiter. "An Extendable Java Framework for Instance Similarities in Ontologies." In: Jan. 2006, pp. 263–269 (cit. on p. 24).

[LN10] Lu Tan, Neng Wang. "Future internet: The Internet of Things". In: *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*. Vol. 5. Aug. 2010, pp. V5-376-V5–380. DOI: 10.1109/ICACTE.2010.5579543 (cit. on p. 21).

[MAF+19] M. Mahdavi, Z. Abedjan, R. Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, N. Tang. "Raha: A Configuration-Free Error Detection System". In: (Jan. 2019) (cit. on pp. 29, 31, 33, 51, 57, 58, 65, 72).

[MF16] I. Markit, Forbes. Nov. 2016. URL: https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/%5C#6a558beb292d (cit. on pp. 15, 21).

[MR17] A. More, V. Raisinghani. "A survey on energy efficient coverage protocols in wireless sensor networks". In: *Journal of King Saud University-Computer and Information Sciences* 29.4 (2017), pp. 428–448 (cit. on p. 70).

[MW14] Z. T. T. Myint, K. K. Win. "Triple patterns extraction for accessing data on ontology". In: *International Journal of Future Computer and Communication* 3.1 (2014), p. 40 (cit. on p. 24).

[PMHM16] C. Pit-Claudel, Z. E. Mariet, R. Harding, S. Madden. "Outlier Detection in Heterogeneous Datasets using Automatic Tuple Expansion". In: 2016 (cit. on pp. 57, 58).

[RCIR17] T. Rekatsinas, X. Chu, I. F. Ilyas, C. Ré. "HoloClean: Holistic Data Repairs with Probabilistic Inference". In: *Proc. VLDB Endow.* 10.11 (Aug. 2017), pp. 1190–1201. ISSN: 2150-8097. DOI: 10.14778/3137628.3137631. URL: https://doi.org/10.14778/3137628.3137631 (cit. on pp. 19, 28, 33, 39, 48–50, 55, 56, 61–63, 65).

[RD00] E. Rahm, H. Do. "Data Cleaning: Problems and Current Approaches". In: *IEEE Data Eng. Bull.* 23 (Jan. 2000), pp. 3–13 (cit. on pp. 16, 17).

[Red98] T. Redman. "The Impact of Poor Data Quality on the Typical Enterprise." In: *Communications of the ACM* 41 (Feb. 1998). DOI: 10.1145/269012.269025 (cit. on p. 16).

[RHW21] Y. Roh, G. Heo, S. E. Whang. “A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective”. In: *IEEE Transactions on Knowledge and Data Engineering* 33.4 (2021), pp. 1328–1347. DOI: 10.1109/TKDE.2019.2946162 (cit. on p. 22).

[SDSB19] J. Seeger, R. A. Deshmukh, V. Sarafov, A. Bröring. “Dynamic IoT Choreographies”. In: *IEEE Pervasive Computing* 18.1 (Jan. 2019), pp. 19–27. ISSN: 1558-2590. DOI: 10.1109/MPRV.2019.2907003 (cit. on pp. 16, 23, 36).

[Sin17] S. Singh. *Industrie 4.0 und das Industrial Internet of Things (IIoT) – eine Einordnung*. Aug. 2017. URL: https://www.industry-of-things.de/industrie-40-und-das-industrial-internet-of-things-iiot-eine-einordnung-a-629710/ (cit. on p. 21).

[SPKN20] P. Schirmer, T. Papenbrock, I. Koumarelas, F. Naumann. “Efficient Discovery of Matching Dependencies”. In: *ACM Trans. Database Syst.* 45.3 (Aug. 2020). ISSN: 0362-5915. DOI: 10.1145/3392778. URL: https://doi.org/10.1145/3392778 (cit. on pp. 23, 39).

[SS13] S. Sagiroglu, D. Sinanc. “Big data: A review”. In: *2013 International Conference on Collaboration Technologies and Systems (CTS)*. 2013, pp. 42–47. DOI: 10.1109/CTS.2013.6567202 (cit. on pp. 15, 16, 22).

[TD96] N. C. S. Technology, S. Division. *Federal Standard 1037C*. General Services Administration Information Technology Service, Aug. 1996 (cit. on p. 22).

[VA18] L. Visengeriyeva, Z. Abedjan. “Metadata-driven error detection”. In: July 2018, pp. 1–12. DOI: 10.1145/3221269.3223028 (cit. on pp. 27, 28, 31, 51).

[WBH19] H. Wang, M. J. Bah, M. Hammad. “Progress in Outlier Detection Techniques: A Survey”. In: *IEEE Access* 7 (2019), pp. 107964–108000. DOI: 10.1109/ACCESS.2019.2932769 (cit. on p. 43).

[YWS17] T. Yu, X. Wang, A. Shami. “Recursive Principal Component Analysis-Based Data Outlier Detection and Sensor Data Aggregation in IoT Systems”. In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 2207–2216. DOI: 10.1109/JIOT.2017.2756025 (cit. on p. 70).

[ZFG+16] M. D. Zio, N. Y. Fursova, T. Gelsema, S. Giessing, U. Guarnera, J. Petrauskienė, L. Quensel, M. Scanu, K. O. ten Bosch, M. van der Loo, K. Walsdorfer. “Methodology for data validation 1.0”. In: 2016 (cit. on pp. 31, 33).

[ZGR20] Y. Zhang, Z. Guo, T. Rekatsinas. “A Statistical Perspective on Discovering Functional Dependencies in Noisy Data”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’20. Portland, OR, USA: Association for Computing Machinery, 2020, pp. 861–876. ISBN: 9781450367356. DOI: 10.1145/3318464.3389749. URL: https://doi.org/10.1145/3318464.3389749 (cit. on pp. 25, 27).

[ZPWV17] L. Zhou, S. Pan, J. Wang, A. V. Vasilakos. “Machine learning on big data: Opportunities and challenges”. In: *Neurocomputing* 237 (2017), pp. 350–361. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.01.026. URL: https://www.sciencedirect.com/science/article/pii/S0925231217300577 (cit. on p. 22).

[ZZL+21] Z. Zheng, L. Zheng, M. A. Langouri, F. Chiang, L. Golab, J. Szlichta. “Discovery and Contextual Data Cleaning with Ontology Functional Dependencies”. In: *CoRR* abs/2105.08105 (2021). arXiv: 2105.08105. URL: https://arxiv.org/abs/2105.08105 (cit. on pp. 23, 25, 26, 28).

All links were last followed on December 9, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature