Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Policy4TDLIoT - Policys für die Topic Description Language

Simon Lehmann

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Dr. rer. nat. Pascal Hirmer, Ana Cristina Franco da Silva, M.Sc.
Beginn am:	14. Juni 2018
Beendet am:	14. Dezember 2018

Kurzfassung

Im Paradigma Internet of Things (IoT), im deutschen *Internet der Dinge*, werden heterogene Geräte über das Internet vernetzt. Diese Geräte enthalten Sensoren und Aktuatoren, um Daten aus ihrer Umgebung zu erfassen und in die Umwelt einzugreifen. Dies ermöglicht die Umsetzung von innovativen Systemen wie *Smart Home*, *Smart City* oder *Smart Factory*. Die Heterogenität der Geräte erschwert es Standardisierungen und einheitliche Metriken zu definieren. Gleichzeitig steigt die Anzahl an vernetzten Geräten und dieser Wachstum wird sich in den kommenden Jahren fortsetzen.

Für diese beiden Probleme wurde die Topic Description Language für die IoT (TDLIoT) entwickelt. *Topics* sind Schnittstellen zwischen Sensoren oder Aktuatoren und Endverbraucher. Sie publizieren die Daten, anhand verschiedener Protokolle (z.B. MQTT, REST) an alle Verbraucher die sich bei ihnen registriert haben. Die TDLIoT ermöglicht es ein Topic mit mehreren Attributen zu beschreiben. Zudem bietet sie ein Katalog an, in dem alle Topic Beschreibungen gesammelt sind und durch Such- und Filterfunktionalitäten gefunden werden können. Die bisherige TDLIoT beinhaltet lediglich die Beschreibung von funktionalen Anforderungen, wie z.B. Datenformat, Datentyp, Zugriffspfad oder Standort.

Das Ziel dieser Arbeit ist es der TDLIoT nichtfunktionale Anforderungen hinzuzufügen, um Topics genauer beschreiben zu können und dem Katalog weitere Filtermöglichkeiten zu bieten. Der Ansatz orientiert sich an WS-Policys der WSDL. Des Weiteren wird die Struktur der TDLIoT durch neue Komponenten und Rollen erweitert. Diese ermöglichen es Policys übersichtlich für die Erstellung eines Topics darzustellen und bieten eine Kontrollstruktur die neue Policys anhand der Anforderungen der TDLIoT überprüft, damit eine hohe Qualität der Policys gewährleistet werden kann. Ein Anbieter eines Topics kann beliebig viele Policys uneingeschränkt definieren. Dadurch können falsche Angaben in den Policys definiert werden. Um dem Nutzer eines Topics eine Sicherheit über den Wahrheitsgehalt eines Topics zu liefern wird es Verifikationen zur Überprüfung der Policy geben. Die Ergebnisse dieser Verifikationen geben jedem Nutzer Rückmeldung, ob das Topic die Angaben aus seinen Policys einhält.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

API Application Programming Interface. 23

AWS IoT Amazon Web Service für IoT. 30

IoT Internet of Things. 15

IoT-DDL IoT Device Description Language. 25

JSON-LD JavaScript Object Notation for Linked Data. 65

MQTT Message Queuing Telemetry Transport. 32

QoS Quality of Service. 24

SLA Service Level Agreements. 24

TD Thing Description. 25

TDLIoT Topic Description Language für die IoT. 15

WoT Web of Things. 25

WSDL Web Services Description Language. 15

1 Einleitung

Das Paradigma Internet of Things (IoT) [LL15], im deutschen *Internet der Dinge*, bezeichnet die globale Vernetzung von Geräten über das Internet von elektrischen und mechanischen Geräten, wobei diese Geräte mit Sensoren und Aktuatoren bestückt sind. Im Laufe der Jahre hat die IoT in verschiedenen Bereichen wie Intelligente Systeme, (z.B. *Smart Cities* [ZBC+14], *Smart Homes* [HHM+17], der Medizin [BS11] oder der Industrie [WMS12] an Relevanz zugenommen und innovative Systeme hervorgebracht. Die momentane Entwicklung führt dazu, dass voraussichtlich bis im Jahr 2020 über 20 Milliarden vernetzte Geräte in der IoT vorhanden sein werden [Gartner17]. Durch diese Menge an verbundenen Geräten können bisher nicht umsetzbare Systeme, beispielsweise Werkzeuge zur Verbesserung von Interaktionen zwischen Mensch und Maschine [KHW+17] oder zur Situationserkennung [HWS+17], entwickelt werden. Für die Umsetzung von neuen IoT-Systemen, welche Sensoren und Aktuatoren verwenden, muss ein Hersteller Informationen über die Geräte erhalten können. Zum einen um in Erfahrung bringen zu können welche Sensoren in diesem Kontext verwendet werden können. Zum anderen, um nähere Information im Bezug auf die jeweiligen Geräten zu erhalten, um vergleichen zu können welche Daten gesendet, welche Aktionen ausgeführt und wie die Sensoren und Aktuatoren angesprochen werden. Ein übliches Verfahren ist, dass ein sogenanntes *Topic* als Zugriffspunkt für Sensordaten oder Aktuatoren dient [CCV12]. Wird beispielsweise die Temperatur durch einen öffentlich zugänglichen Sensor gemessen, hat dieser in den wenigsten Fällen die Speicher- und Rechenkapazität, um Messwerte für eine große Anzahl an Nutzern bereitzustellen. Durch ein Topic wird die Funktionalität des Verbindungsaufbaus für den Sensor abgenommen. Das heißt, dass jeder neue Messwert vom Sensor zum Topic gesendet wird und dieser publiziert die Daten an alle registrierten Nutzer. So wird eine Abgrenzung zwischen Sensor und Endnutzer geschaffen, die für mehr Sicherheit und Flexibilität in der Kommunikation sorgt. Damit Topics global auffindbar und zentral zugänglich sind, wurde die Topic Description Language für die IoT (TDLIoT) [FHB+18] entwickelt. Die TDLIoT beschreibt ein Topic und stellt diese Beschreibungen gesammelt in einem Katalog zur Verfügung. Die TDLIoT ist durch fünf Anforderungen definiert. Das sind Generizität, technologische Unabhängigkeit, Abgeschlossenheit, Leichtgewichtigkeit und Erweiterbarkeit. Die Beschreibung eines Topics von Franco da Silva et al. [FHB+18] bezieht sich auf die funktionalen Anforderungen, das bedeutet, es wird beschrieben inwiefern der Zugriff der Daten ermöglicht wird und wie diese Daten aussehen.

Das Ziel dieser Arbeit ist nichtfunktionale Anforderungen der TDLIoT hinzuzufügen, welche in diesem Kontext Policy genannt wird. Dabei orientiert sich die Erweiterung an Web Services Description Language (WSDL) [WCL+05], welche bereits durch WS-Policy

nichtfunktionale Anforderungen eingebunden hat. Eine nichtfunktionale Anforderung an eine Software, beschreibt wie eine Funktion des Systems umgesetzt wird, im Gegensatz dazu beschreibt die funktionale Anforderung was die Funktionalität in der Software macht [TD95]. Bezogen auf ein Topic sind nichtfunktionale Anforderungen beispielsweise die Aussage darüber wie Werte gespeichert oder übermittelt werden und inwiefern das Topic zur Verfügung steht. Das können Anforderungen bezogen auf das Topic sein (z.B. Kosten, Zuverlässigkeit, Benutzbarkeit [CNYM12]) oder nichtfunktionale Anforderungen bezogen auf die Nachrichten die über das Topic publiziert werden (z.B. Genauigkeit, Abweichung, Invervall). Dazu kommt, dass in der Regel nichtfunktionale Anforderungen schwieriger zu testen sind als funktionale Anforderungen, was für die Erweiterung der TDLIoT eine zusätzliche Rolle spielt.

Zu Beginn muss durch die Einarbeitung in das Thema der nichtfunktionalen Anforderungen für die IoT ein Modell einer Policy definiert werden. Durch das spätere Einbeziehen dieses Modells in die TDLIoT müssen die bisherigen Anforderungen an die TDLIoT weiterhin eingehalten werden. Gleichzeitig müssen unterschiedliche Arten von nichtfunktionalen Anforderungen abgebildet werden können. Das kann zum Beispiel die Genauigkeit eines Temperatursensor sein, welcher angibt auf wie viele Nachkommestellen gemessen wird oder die Kosten, die Anfallen, um eine Schranke (einen Aktuator) im Parkhaus bedienen zu können. Nach der Modellierung muss die TDLIoT auf konzeptioneller Ebene durch Policys erweitert werden. Dafür soll ein Ablauf entwickelt werden, der einem Ersteller einer TDLIoT, genannt *Topic Provider*, erlaubt eine Sammlung von bereits definierten Policys zu durchsuchen und die für seine Anwendungsfälle passenden Policys seiner Beschreibung hinzuzufügen. Falls der Topic Provider keine Policy findet, die seine nichtfunktionale Anforderung beschreibt, soll er die Möglichkeit haben selbst eine Policy zu definieren, die wiederum nach der Erstellung in die Sammlung aufgenommen wird, um für andere Topic Provider verwendbar zu sein. Ein Problem wird sein, dass jeder Topic Provider durch die Policys Aussagen über sein Topic treffen kann, die nicht der Wahrheit entsprechen. Deshalb sollen *Topic Consumer*, welche diejenigen sind, die Daten eines Topics verwendet möchten, die Möglichkeit haben eine Verifizierung für eine Policy auszuführen. Die Ergebnisse solcher Verifikationen werden für andere Topic Consumer festgehalten, damit sie sehen ob die Aussagen der Policys eingehalten wurden und gegebenfalls neue Verifikationen starten können, falls die letzte Überprüfung zu lange her ist. Manche nichtfunktionale Anforderungen, wie die Abweichung oder Genauigkeit eines Sensors, können nicht überprüft werden, das sollen dem Topic Consumer genauso mitgeteilt werden. Zum Schluss soll die Verifizierung der Policys im Prototypen der TDLIoT[1] implementiert und die Erweiterung durch Policys integriert werden. Die Erweiterung von Policys bezieht sich auf die Attribute des bisherigen Modells der TDLIoT und auf die Integration des Ablaufs zur Findung und dem Hinzufügen von neuen Policys. Durch die Änderung des Datenmodells muss zudem die Such- und Filterfunktionalität der Topic Beschreibungen angepasst werden, damit abhängig von Policys, Topics in der TDLIoT gefiltert werden können.

[1] https://github.com/IPVS-AS/TDLIoT

Diese Arbeit ist wie folgt aufgebaut: In Kapitel 2 werden die wichtigsten Grundlagen dieser Arbeit vorgestellt. Dazu gehören die Beschreibung der Begriffe IoT, TDLIoT und Policys. Kapitel 3 zeigt verwandte Arbeiten auf und wie sie sich zu dieser Arbeit unterscheiden. Die Konzeption der Erweiterung der TDLIoT durch Policys wird in Kapitel 4 erläutert. Darauf aufbauend ist in Kapitel 5 die prototypische Implementierung der Konzeption beschrieben. Zum Schluss wird in Kapitel 6 die Arbeit zusammenfassend erläutert und die Vorteile dieser Umsetzung aufgezeigt. Zudem gibt es einen Ausblick auf zukünftige Arbeiten für die TDLIoT.

2 Grundlagen

In diesem Kapitel werden Grundlagen dieser Arbeit aufgezeigt. Diese sind nötig, um die einzelnen Thematiken und ihre Zusammenhänge zu verstehen. Dazu gehört das Thema IoT, hier wird speziell auf Topics eingegangen da sie für diese Arbeit eine essentielle Rolle spielen. Des Weiteren wird die TDLIoT vorgestellt, ein Ansatz, um Topics in der IoT zu beschreiben und für Verbraucher gesammelt auffindbar zu machen. Zuletzt wird der Begriff *Policy* beschrieben, da die TDLIoT im Rahmen dieser Masterarbeit durch nichtfunktionale Anforderungen erweitert wird.

2.1 Internet of Things (IoT)

Nach Atzori et al. [AIM10] ist die Idee von IoT die Vernetzung physischer und virtueller Geräte über das Internet. Die Geräte in der IoT sind heterogen und können deshalb kleinste Messsensoren sein oder die Komplexität eines heutigen Computersystems haben. Sie sammeln über Sensoren Daten und stellen diese für andere Dienste zur Verfügung oder ermöglichen über Aktuatoren den Eingriff in ihre Umgebung. Im Bereich der Wissenschaft zeigt Abbildung 2.1 die steigende Anzahl an Publikationen in der IoT in den vergangen Jahren, laut der Webseite *Dimensions*[1]. Diese Vernetzung und große Anzahl an heterogenen Geräten ermöglicht innovative System aber birgt auch Probleme. Gerade im alltäglichen Leben kann IoT viele positive Veränderungen mit sich bringen. Das kann das vereinfachte Finden eines Arztes im Urlaub sein [BBKM15] oder eine App, die anzeigt, welche Parkplätze in der Stadt zur Verfügung stehen [GBMP13]. Gleichzeitig können durch die Vernetzung neue Probleme entstehen. Da IoT-Geräte von ihrer Speicher- und Rechenkapazität sehr klein sein können sind sie Schwachstellen für die Sicherheit in einem System. Gerade wenn Geräte Daten erheben, die einem sehr hohen Datenschutz unterliegen, muss auf Sicherheit und Verschlüsselung besonderes geachtet werden. Genauso schwierig ist es Standards für die heterogenen Geräte zu definieren, da die Abbildung vieler verschiedener Systeme und Geräte komplexe Anforderungen erzeugt.

[1] https://app.dimensions.ai/discover/publication

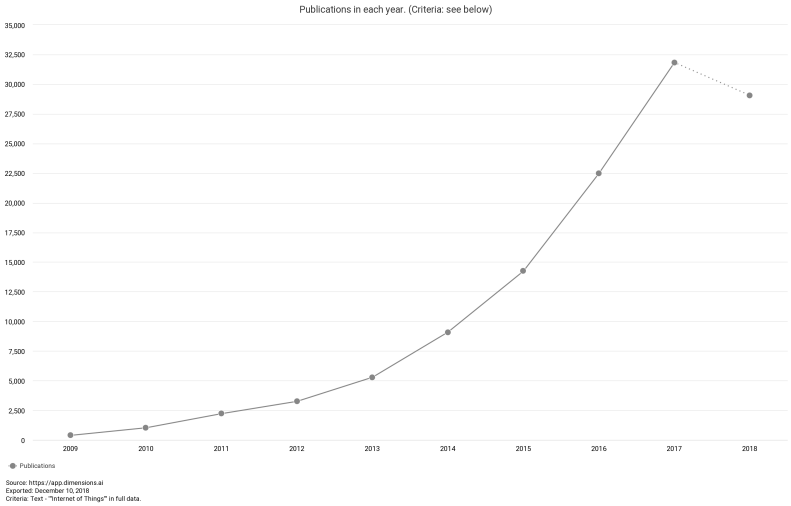

Abbildung 2.1: Publikationen in den letzten Jahren im Bereich IoT laut der Webseite *Dimensions*

2.1.1 Topics in der IoT

Für diese Arbeit spielt die Schnittstelle zwischen Verbraucher und Sensor oder Aktuator eine wichtige Rolle. Sie wird *Topic* genannt und publiziert Sensorwerte an alle angemeldeten Verbraucher oder stellt Zugriff auf Aktuatoren bereit [CCV12]. Ein Topic ist das Gegenteil einer *message queue*. Der Unterschied liegt darin, dass die Daten nicht gespeichert, sondern vom Topic an alle angemeldeten Systeme weitergeleitet werden. Falls zu diesem Zeitpunkt niemand angemeldet ist, werden die Daten nicht gespeichert aber dennoch publiziert. Somit gehen, im Gegenteil zu einer *message queue*, die Daten verloren, wenn es keinen Abnehmer dafür gibt. Der Zugriff auf ein Topic erfolgt über einen Pfad, welcher wie in Abbildung 2.2 dargestellt, aus Topic Level und Separatoren besteht. In diesem Beispiel ist es der Pfad für das Topic des Temperatursensors in einem Pool. Mithilfe der Separatoren sind die einzelnen Topic Level untergliedert. Zudem ist es möglich Gruppen von Topics abzurufen, falls in einem Level mehrere Topics vorhanden sind. Durch diese Unterteilung können Topics flexibel und mit einer leichtgewichtigen Syntax angesprochen werden.

Abbildung 2.2: Beispielhafter Pfad eines Topics

Für die TDLIoT und somit auch für die Erweiterung der TDLIoT durch Policys ist ein Topic eine Entität, die Daten senden und empfangen kann. Dies kann mit verschiedenen Protokollen umgesetzt werden (z.B. MQTT, REST) die zum einen ein *publish-subscribe* Modell anbieten oder zum anderen das *request-response* Modell.

Ein publish-subscribe Modell entkoppelt die Verbindung zwischen Sender und Empfänger [HBS+16]. In diesem Kontext wird der Sender *Publisher* und der Empfänger *Subscriber* genannt. Zwischen ihnen wird eine weitere Komponente, der *Broker* platziert, die die Nachrichtübermittlung verwaltet. Sie leitet alle eingehenden Nachrichten des Publishers an die angemeldeten Subscriber weiter. Dadurch muss ein Publisher seine Subscriber nicht kennen, sondern nur den Pfad zu seinem Broker, um die Daten zu veröffentlichen. Genauso müssen Subscriber nur die Verbindung zum Broker herstellen. Der Broker selbst leitet die richtigen Nachrichten weiter. Somit kann für unterschiedliche Daten der gleiche Broker verwendet werden.

Ein request-response Modell repräsentiert das klassische Server-Client-Modell bei dem der Client eine Anfrage zur Erhaltung von Daten sendet und der Server oder Service darauf reagiert. Bei einer akzeptierten Anfrage werden die Daten übermittelt ansonsten wird eine Fehlerrückmeldung gesendet. Der *Request* erfolgt somit durch den Client, die *Response* vom Server. Bei diesem Modell muss der Client wissen wie der Server anzusprechen ist und welche Daten dieser bereitstellt. Zudem ist die zeitliche Abfolge vordefiniert, da nach einer Anfrage immer eine Antwort folgt muss, bevor eine weitere Anfrage gesendet werden kann.

2.2 Topic Description Language für die IoT

Die TDLIoT wurde von Franco da Silva et al. [FHB+18] entwickelt. Sie beziehen sich auf das Problem, dass an mancher Stelle die Entwicklung von innovativer Software im Bereich IoT nicht umgesetzt werden kann. Das liegt daran, dass das Wissen darüber, welche Sensoren und Aktuatoren angeboten werden und verfügbar sind, nicht gebündelt existiert. Zudem gibt es keine standardisierte Beschreibung von Topics. Dabei ist es notwendig zu verstehen wie ein Topic angesprochen werden kann, welche Daten der Sensoren zur Verfügung stellen und welche Aktionen ein Aktuator ausführt. Um das zu lösen wurde die TDLIoT entwickelt. Hierfür sind fünf Anforderungen an die TDLIoT definiert worden, um die Vielfalt an Anwendungsfällen, die es im IoT-Bereich gibt, abzudecken.

Generisch Es gibt in der IoT keine Definition, wie ein Gerät aufgebaut sein muss. Deswegen sind die Geräte heterogen und enthalten verschiedene Arten von Sensoren und Aktuatoren. Die TDLIoT muss diese Individualität zulassen und dementsprechend generisch sein.

Technologische Unabhängigkeit Da in der IoT unterschiedliche Technologien verwendet werden muss es eine technologische Unabhängigkeit geben. Dies bezieht sich auf Protokolle, *Middleware* oder Netwerkschnittstellen.

Abgeschlossen Die TDLIoT soll in sich abgeschlossen sein. Das heißt, dass alle Informationen zum Topic in der TDLIoT enthalten sein müssen, ohne Zugriff auf externe Quellen zu haben. Dadurch kann eine effiziente Such- und Filterfunktionalität gewährleistet werden.

Leichtgewichtig Viele Geräte der IoT sind in ihrer Prozessleistung und Speicherplatz begrenzt. Gleichzeitig sollte die Einarbeitung in die Sprache so leicht wie möglich gemacht werden. Daraus folgt, dass die TDLIoT eine kompakte und einfache Beschreibung seins soll.

Erweiterbar Gerade weil die TDLIoT in der Entwicklung ist, aber auch weil das Feld der IoT sich schnell wandeln kann, soll die TDLIoT erweiterbar bleiben, um alle Eventualitäten abdecken zu können. Als Beispiel kann diese Arbeit verwendet werden, da die TDLIoT durch Policys erweitert wird.

Der Aufbau der TDLIoT ist in Abbildung 2.3 dargestellt. Es ist ein *Entity-Relation* Modell mit der Struktur nach Chen [Che76]. In dieser Darstellungen werden die Zusammenhänge zwischen den einzelnen Attributen aufgezeigt. Jedes Attribut, bis auf *hardware type* und *unit*, muss genau einmal in der Datenstruktur eines Topics vorkommen. *Hardware type* und *unit* sind optional und müssen nicht enthalten sein. Im Folgenden wird jedes Attribut genauer erläutert:

- **data type** Der Datentyp der angebotenen Werte.
- **hardware type** (*optional*) Der Art des Sensors oder Aktuators.
- **location** Der Ort der Hardware basierend aus dem Ortstyp, welches Breiten- und Längengrad oder Städtename sein kann, und die dazugehörige Information in *location value*.
- **message format** Das Format der Nachricht, die das Topic anbietet. Beispielsweise JSON, YAML oder XML.
- **message structure** Der strukturelle Aufbau der Daten die über das Topic erhalten werden.
- **middleware edtpoint** Die Adresse um das Topic auf dem Server oder anderen Endpunkt im Netz anzusprechen.
- **owner** Der Name des Topicanbieters.
- **path** Der Pfad des Topics.

- **protocol** Das Protokoll, mit dem die Nachricht übertragen wird. Beispielweise MQTT oder HTTP.
- **topic type** Der Typ des Topics, bei MQTT wäre das *subscribe*, bei HTTP wäre es *request*.
- **unit** (*optional*) Die Einheit der Werte wie zum Beispiel Celsius als Temperatureinheit.

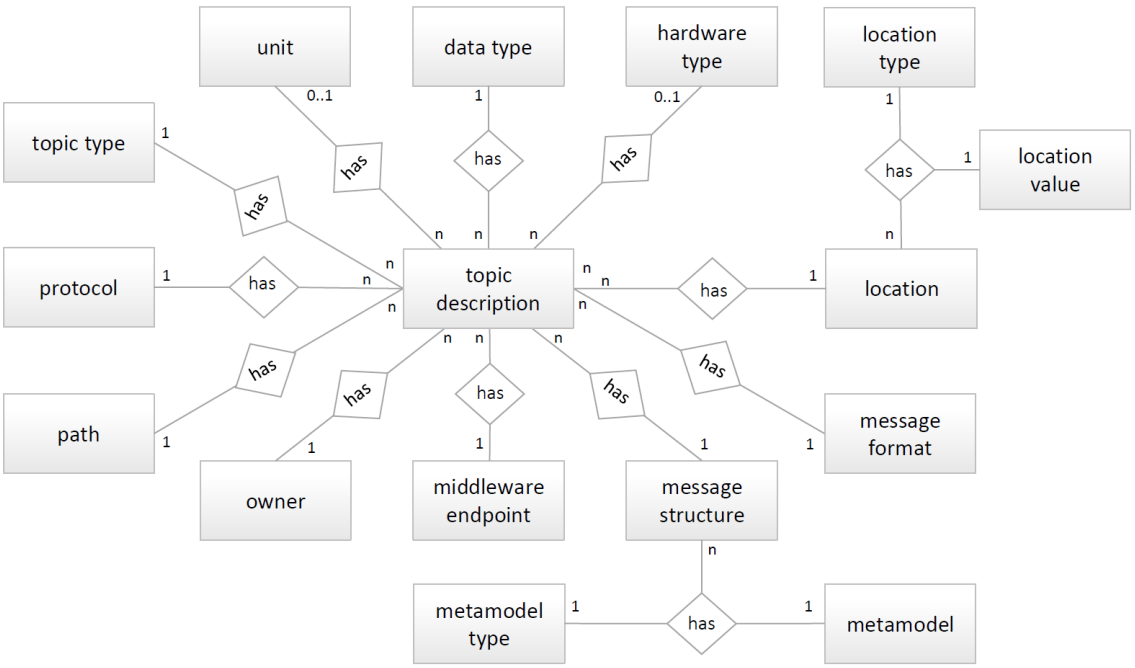

Abbildung 2.3: Struktureller Aufbau der TDLIoT

Damit das Konzept einer einheitlichen Beschreibung von Topics umgesetzt werden kann, wurde zudem ein Katalog mit REST Application Programming Interface (API) entwickelt. Der *Topic Catalog* enthält alle bereits erstellen Topic Beschreibungen und bietet über die Webseite einen Überblick über sie. Mithilfe der Such- und Filterfunktionalität kann nach Topics gefiltert werden. Dazu gehört eine REST Schnittstelle, die erlaubt automatisiert und eingebettet in anderen Systemen Zugriff auf den Katalog zu haben. Zusätzlich kann bei beiden, der Webseite und der REST Schnittstelle, Topics hinzugefügt, gelöscht und verändert werden. Zur Nutzung des Topic Catalog wurden zwei Rollen definiert. Die erste ist der *Topic Provider*. Er kann neue Topic Beschreibungen von seinen Topics erstellen und sie im Katalog veröffentlichen. Zudem wird über den Katalog die Möglichkeit angeboten bisherige Topic Beschreibung anzupassen und zu entfernen. Die zweite Rolle ist der *Topic Consumer*. Er ist auf der Suche nach Daten in der IoT und kann über den Katalog sein gewünschten Topics, über die Such- und Filterfunktionalität, finden und die Verbindung mithilfe des *middleware endpoints* und dem *path* aufbauen.

2.3 Policy

Nach Thayer et al. [TD95] ist eine nichtfunktionale Anforderung eine Anforderung an die Software, die beschreibt, wie eine Funktionalität in der Software umgesetzt wird und nicht was diese Funktionalität in der Software macht. Das kann beispielsweise die Beschreibung der Zuverlässigkeit, der Kosten oder der Wartbarkeit eines Softwaresystems sein. Die Policys für die TDLIoT orientieren sich an der Definition und den Anforderungen der WS-Policys für die WSDL [WCL+05]. Sie bilden nichtfunktionale Anforderung ab, die aus den Bereichen der Quality of Service (QoS) oder Service Level Agreements (SLA) stammen können, und beschreiben Anforderungen an Topics in der IoT. Aus dem Szenario von Franco da Silva et al. [FHB+18], in dem eine App anzeigt, welche freien Parkplätze in der Umgebung vorhanden sind können verschiedene Policys für Topics beispielhaft definiert werden. Das kann für die Parkplatzsensoren die Zuverlässigkeit sein, da Ausfälle für den Topic Consumer zu Kosten oder Wertverlusten im Bezug auf seine Applikation führen können. Zum anderen könnte es die Aktualität der Werte sein, die von den Sensoren erfasst werden, da ein Fahrer, der auf der Suche nach einem Parkplatz ist, eine Echtzeitanwendung benötigt. Policys sind eine wichtige Angabe für die Gewährleistung der Qualität eines Dienstes. Die Schwierigkeit daran ist die Überprüfung, ob die getroffenen Aussagen einer Policy eingehalten werden. Als Beispiel kann die Abweichung eines Sensors nicht überprüft werden, da der Fehler der Messung in der Hardware steckt und dadurch die Software dahinter keine Verifizierung des Wertes ohne zusätzliche Hardware durchführen kann.

3 Verwandte Arbeiten

Für die Weiterentwicklung der IoT ist es wichtig, Standards zur Beschreibung von Sensoren und Aktuatoren zu definieren. Die TDLIoT [FHB+18] ist ein Beispiel für derartige Forschung. Sie beschreibt Topics, die zwischen Endnutzer und Sensoren oder Aktuatoren platziert sind. In diesem Kapitel zur TDLIoT verwandte Arbeiten vorgestellt und die Unterschiede erläutert. Des Weiteren werden im Bezug auf nichtfunktionale Attribute und Anforderungen verschiedene Herausforderungen aufgezeigt die in dieser Arbeit für die Konzeptionsentwicklung und Implementierung der angestrebten Policys eine wichtige Rolle spielen.

3.1 Beschreibungssprachen in der IoT

Khaled et al. [KHLL18] haben eine für Menschen und Maschinen lesbare Beschreibungssprache für Geräte in der IoT, mit dem Namen IoT Device Description Language (IoT-DDL), entwickelt. Zudem wird auf Basis der IoT-DDL eine Architektur vorgestellt, die neue Dienste zur Interaktion mit den Geräten bereitstellt. Sie erlaubt, dass Geräte sich gegenseitig entdecken und die jeweiligen Dienste, Fähigkeiten und Entitäten untereinander geteilt werden können, wobei Standards im Bereich Sicherheit, Kommunikation, Geräteverwaltung und Microservices genutzt werden. Es existieren diverse Unterschiede zur TDLIoT. Zum einen wird in der IoT-DDL ein Gerät beschrieben und in der TDLIoT ein Topic, was in der Modellierung eines Objektes zu unterschiedlicher Priorisierung von Attributen und Umfang eines Modells führt. Ein Gerät kann aus verschiedenen Sensoren und Aktuatoren bestehen, wie das Beispiel einer Kaffeemaschine [KHLL18], was die Komplexität der TDLIoT übersteigt. Zum anderen muss die für die IoT-DDL bereitgestellte Architektur befolgt werden. Diese kann beispielsweise durch eine *Firmware* auf die Geräte bespielt werden. Im Gegenteil dazu greift die TDLIoT nicht in die Kommunikation über das Netzwerk ein, sondern ist lediglich eine Sprache zur Beschreibung von Topics. Die TDLIoT stellt darüber hinaus einen Katalog mit existierenden Topics zur Verfügung. Zudem ist die IoT-DDL eine XML-basierte Beschreibungssprache, die TDLIoT wiederum ist leichtgewichtiger und in einem kompakteren Datenformat wie JSON oder YAML definiert.

Das Web of Things (WoT), entwickelt von W3C, hat das Ziel der Vernetzung in der IoT. Ein Teil davon ist die glsWoT glsTD [W3C18], die eine Beschreibung von Geräten und Diensten in der IoT darstellt. So kann ein *Thing* aus verschiedenen Eigenschaften, Aktionen und Events bestehen. Eine Thing Description (TD) wird im JSON Format abgebildet und

an einem definierten Schema festgelegt. Jedes *Thing* enthält eine eigene Beschreibung und kann über die WoT API als Schnittstelle zur Entdeckung und Interaktion untereinander genutzt werden. Im Vergleich zur TDLIoT, die eine Beschreibung von Topics ist, können mit der WoT TD komplexere Geräte und Dienste abgebildet werden. Die TD ist in das System der WoT integriert und wird in dieser Umgebung verwendet. Dagegen ist die TDLIoT eine systemunabhängige Beschreibung die überall integriert werden kann und Topics in ihrer Gesamtheit definiert. Das Ziel der TD ist die Fähigkeit der Vernetzung zwischen den Geräten und Diensten, die Interaktionen und das Finden von anderen *Things* erlaubt. Auch hier hat die TDLIoT ein anderes Ziel, da sie als Sammelstelle für Topic Beschreibung fungiert und diese in einem Katalog gebündelt darstellt, unabhängig von Schnittstellen und Kommunikation zwischen Topics oder Geräten.

3.2 Policys

Zhang et al. [ZCW+14] stellen Herausforderungen bezüglich Sicherheit von IoT-Geräten durch ihre Homogenität sowie deren hohen Anzahl in der IoT dar. Daraus entstehen Sicherheitsprobleme in zwei Schwerpunkten, zum einen die Vielfalt an unterschiedlichen Arten von Geräten und die Kommunikation untereinander. Sie zeigen auf, dass, im Vergleich zu traditionellen Systemen, die von Windows Betriebssystemen dominiert werden, der Fokus nicht auf eine Systemarchitektur oder Technologie gelegt sein darf. Geräte in der IoT sind beispielsweise Umgebungssensor, intelligente Küchengeräte oder tragbare Uhren, die alle mit unterschiedlichen Betriebssystemen, Speicherverfahren und Speicherkapazitäten bestückt sein können. Dadurch sind sie für Absicherungen und Verschlüsselungen unterschiedlich kompatibel und können gegebenfalls rein von der Speicherkapazität nur eine beschränkte Sicherheit bieten. Im Vergleich dazu können Daten die sie messen und speichern Alltagsgewohnheiten und Vorlieben der Menschen sein welche sehr hohen Datenschutzbestimmungen unterliegen. Das gesamte Thema ist demnach mit den bisherigen Restriktionen sehr komplex und schwierig zu bewältigen.

Für die Erweiterung der TDLIoT durch Policys sollen Sicherheits- und Verschlüsselungsarten eines Topics abgebildet werden können. Die Artenvielfalt und hohe Anzahl von Geräten zeigt wie unterschiedlich ein Topic sein kann. Genauso kann eine Relation zwischen den Herausforderungen an die Sicherheit in der IoT und des Abbildens einer Policy, die die Sicherheit definiert, hergestellt werden. Erweiterbarkeit wird für Policys in der TDLIoT eine wichtige Rolle sein, da in diesem Bereich stetig neue Systeme und Technologien entwickelt werden.

Der Begriff nichtfunktionale Anforderungen hat im Laufe der Jahre unterschiedlichste Definition erhalten. Glinz [Gli07] stellt dar inwiefern der Kontext einer nichtfunktionale Anforderung die Definition beeinflusst. Er zeigt auf, dass durch Klassifizierung der Anforderungen eine eindeutigere Definition getroffen werden kann. Zudem ermöglicht es die Einteilung der Verifikation einer nichtfunktionalen Anforderung treffender zu definieren. Tabelle 3.1 zeigt für die verschiedenen Repräsentationen einer nichtfunktionale Anforderung

die Art ihrer möglichen Verifikation. Da eine Aufgabenstellung für die Erweiterung der TDLIoT durch Policys die Verifikation, zur Sicherstellung des Wahrheitsgehaltes eines Topics ist, gibt diese Auflistung von Verifikationsarten einen Überblick in welcher Form Ergebnisse geliefert werden können. Sie zeigt, dass es für qualitative Anforderungen keine direkte Verifizierung gibt und dies Teil der TDLIoT sein muss, damit ein Topic Consumer weiß, dass gewisse Policys nicht verifiziert werden können. Gleichzeitig gibt es messbare und überprüfbare nichtfunktionale Anforderungen, die im Katalog der TDLIoT mit ihrem Überprüfungsergebnis dargestellt werden sollen.

Repräsentation	**Arten der Verifikation**
in Betrieb	Review, Tests oder Formale Verifikation
Quantitativ	Messung
Qualitativ	keine direkte Verifikation möglich
Deklarativ	Review

Tabelle 3.1: Klassifizierung der Anforderungen nach Arten der Verifikation [Gli07]

4 Policy4TDLIoT - Konzeptionelle Lösung

Die Erweiterung der TDLIoT um *Policys* beinhaltet neue Anforderungen für das Gesamtkonzept der TDLIoT. Dazu gehören (1) die Modellierung und das Design der Policys, um nichtfunktionale Anforderungen definieren zu können, (2) die Konzeption der TDLIoT-Erweiterung durch die neuen Policys, (3) die Integration der Policys in die TDLIoT und (4) das Konzept einer Verifikation von Policys. Diese Verifikation soll sicherstellen, dass die Policy eines *Topics* das Einhält, was sie besagt. Wenn beispielsweise eine Policy verlangt, dass vor dem Zugriff auf ein Topic eine Authentifizierung benötigt wird, darf es nicht möglich sein, ohne Authentifizierung die Daten des Topics zu erhalten. Im folgenden Kapitel werden diese Anforderungen in ihre Teilaspekte zerlegt und eine konzeptionelle Lösung dargestellt. Das beinhaltet die Modellierung der Policy und die Anpassung der Rollen und Komponenten der TDLIoT. Bereits vorhandene Komponenten müssen angepasst werden, um das Konzept von Policy integrieren zu können. Zusätzlich müssen neue Bestandteile der TDLIoT konzipiert werden. Dazu gehört ein Mechanismus um Policys für *Topic Provider* und *Topic Consumer* zu definieren, damit sie herausfinden können welche Policys bereits existieren. Es soll klar hervorgehen für welchen Anwendungsfall eine Policy angewendet wird, damit nicht mehrere Policys den selben Zweck erfüllen. Darüber hinaus wird eine Kontrollstruktur benötigt, die gewährleistet, dass die nötige Qualität der Beschreibung von Policys vorhanden ist. Am Ende dieses Kapitels werden die definierten Anforderungen evaluiert.

4.1 Anforderungsanalyse

Um die Konzeption von Policys für die TDLIoT umsetzen zu können muss analysiert werden, wie diese definiert sein müssen und welchen Anforderungen existieren. Die Anforderungen an die Policys werden dabei aus realen Szenarien abgeleitet. Die Anforderungen an die TDLIoT: Generizität, technologische Unabhängigkeit, Abgeschlossenheit, Leichtgewichtigkeit und Erweiterbarkeit bleiben dabei bestehen [FHB+18]. Dennoch werden weitere Anforderungen für die Erweiterung der TDLIoT von Policys aufgestellt. Dabei handelt es sich um Anforderungen die sich auf das Abbilden der unterschiedlichen Arten von Policys beziehen, um die Komplexität und Relationen zwischen Policys darstellen zu können. Gleichzeitig aber auch die Konzeption für das Erstellen, Ändern und Finden von Policys. Ein weiterer Punkt ist die Qualität und Eindeutigkeit einer Policy Beschreibung,

die durch eine Person oder Automation überprüft und gewährleistet werden muss. Die im Folgenden beispielhaft aufgeführten sieben *Policy Types* repräsentieren nicht die komplette Vielfalt von möglichen Policys, geben jedoch einen Hinweis wie diese angewendet werden können. Dadurch wird außerdem deutlich wie unterschiedlich der Anwendungsfall einer Policy sein kann.

Interval Der Policy Type Intervall definiert wie oft ein neuer Wert in einem Topic veröffentlicht wird. Amazon Web Service für IoT (AWS IoT) [Ser18] benutzt diese Angabe, um aufzuzeigen wie viele Daten pro Zeiteinheit an ein Topic gesendet wird. Das Intervall ist bei AWS IoT in enger Kombination zu den Kosten, da mit steigender Datenmenge auch die Kosten steigen. Diese enge Verknüpfung würde die TDLIoT zu sehr einschränken, da Kosten unabhängig des Datenintervalls angesetzten werden können. Um genau definieren zu können wie viel Werte pro Intervall publiziert wird benötigt dieser Policy Type zwei Angaben. Zum einen die Länge des Intervalls, dies kann eine Sekunde, 20 Millisekunden oder mehrere Tage sein. Zum anderen der Typ der Daten, d.h ob es sich um numerische Werte, textueller Inhalt oder eine Zusammenstellung von mehreren Werten handelt. Listing 4.1 stellt in einer abstrakten Form ein mögliches Beispiel dieses Policy Types dar. Dabei handelt es sich um einen Intervall von 200 Millisekunden, in diesem Rhythmus werden Integer Werte des Sensors publiziert.

Listing 4.1 Beispiel eines Policy Types des Intervalls

```
1 Interval:
2     "interval value": 200,
3     "unit":           "milliseconds",
4     "data type":      "integer"
```

Pricing Kosten können aus unterschiedlichen Gründen in der IoT für einen Topic Consumer auftreten. In der AWS IoT [Ser18] beziehen sie sich hauptsächlich auf den Bandbreitenbedarf. Dieser ist gekoppelt mit Aufrufen an die bereitgestellten Topics. Für die TDLIoT muss dieser Policy Type so generisch gehalten sein, damit jede Kombination aus Kosten und einem beliebigen anderen Policy Type abgedeckt ist. Daraus lassen sich drei Angaben ableiten. Zum einen muss angegeben werden, wie oft für den Zugriff auf ein Topic bezahlt werden muss. Das kann eine einmalige oder wiederkehrende Zahlung in unterschiedlichen Zyklen sein. Zum anderen muss die Höhe der Kosten die der Nutzer zu zahlen hat angegeben werden. Darüber hinaus muss für die Kosten klar definiert sein in welcher Währung bezahlt wird. Listing 4.2 repräsentiert ein beispielhaftes Szenario dieses Policy Types. Die Policy besagt, dass für das Anmelden, an das Topic, Kosten entstehen. Diese Kosten müssen einmalig in Hohe von 0,99 Euro an den Betreiber des Topics bezahlt werden.

Listing 4.2 Beispiel eines Policy Types der Kosten

```
Pricing:
    "cost":       0.99,
    "cost_cycle": "once",
    "currency":   "euro"
```

Availability Die Verfügbarkeit definiert die maximale Ausfallzeit (z.B. durch Wartung, Abstürzen) eines Services in einem definierten Zeitraum (z.B. 10 Stunden / Jahr). Dies wird in sogenannten SLA [MAP+18] zwischen Servicekonsumenten und Serviceanbietern vereinbart. Überschreitungen der Ausfallzeit sind in diesem ebenfalls geregelt. Cleland-Huang et al. [CSZS07] zeigen auf, dass die Verfügbarkeit eine nichtfunktionale Anforderung ist und dementsprechend in der TDLIoT als Policy Type beschrieben werden muss. Da die Verfügbarkeit oftmals in Prozent angegeben wird sind hier zwei Angabe nötig. Dies ist die durchschnittliche Zeit, die das Topic verfügbar ist oder die prozentuale Ausfallzeit des Topics. Listing 4.3 ist ein Beispiel für diesen Policy Type. Es sagt aus, dass das Topic 99,99% der Zeit verfügbar ist. Das Jahr besteht aus 8760 Stunden, davon würde das Topic 87,6 Stunden keine Werte liefern.

Listing 4.3 Beispiel eines Policy Types der Verfügbarkeit

```
Availability:
    "percentage":  99.99,
    "state": "available"
```

Authentication Die Sicherheit eines Topics setzt sich aus verschiedenen Teilbereichen zusammen. Dazu gehört die Verschlüsselung von Nachrcihten, die Sicherheit vor Angriffen auf das Topic und die Zugriffsbeschränkung durch eine Authentifizierung. Wir betrachten nur den Bereich der Authentifizierung die unter Cleland-Huang et al. [CSZS07] zu den nichtfunktionalen Anforderungen gehört. Es gibt verschiedene Arten von Authentifizierungen, das kann sich um einen Schlüssel, also einem einzelnen Ausdruck, handeln oder einer Kombination aus Zugriffsidentifizierung, in der Regel Benutzername und Passwort. Der Policy Type Authentifizierung muss somit alle möglichen Arten von Zugriffsberechtigungen ermöglichen. Listing 4.4 zeigt ein abstraktes Beispiel dieses Policy Types. Sie besagt, dass der Zugriff auf die Daten des Topics eine Authentifizierung benötigt. Dieser erfolgt über ein Passwort und Benutzername.

Listing 4.4 Beispiel eines Policy Types der Authentifizierung

```
Authentication:
    "credentials": "username and password"
```

Accuracy Jeder Sensor liefert Sensordaten in einer durch die Hardware limitierten Genauigkeit. Anhand des Datenblattes eines DS18B20 Temperatursensor [Int18] kann aufgezeigt werden, dass die Genauigkeit in einer realen Umgebung variiert und woran diese auszumachen ist. Im Datenblatt wird die Temperatur auf drei Nachkommastellen limitiert. Die Genauigkeit, als Policy Typ, braucht dementsprechend eine Angabe des minimalen Wertes, um zeigen zu können welche Genauigkeit der Sensor bietet. Listing 4.5 ist ein Beispiel für die Attribute dieses Policy Types. Sie besagt, dass die Werte, die vom Topic publiziert werden, eine maximale Genauigkeit von drei Nachkommastellen besitzt.

Listing 4.5 Beispiel eines Policy Types der Genauigkeit

```
1 Accuracy:
2     "accuracy":  0.001
```

Deviation Die Abweichung eines Sensors ist, wie bei der Genauigkeit, abhängig von den Bauteilen aus denen der Sensor besteht. Im Datenblatt des DS18B20 [Int18] ist dargestellt wie eine Abweichung definiert wird und in welchem Maße sie den Sensorwert beeinflusst. Sie wird als prozentuale Abweichung, die den Wert nach oben und unten verändern kann, festgehalten. Somit ist die Ungenauigkeit eines Sensors von Genauigkeit und Abweichung abhängig. Da die prozentuale Angabe der Abweichung alle möglichen Szenarien abdeckt, reicht es wenn der Policy Type der Abweichung mit dieser Angabe definiert wird. Listing 4.6 stellt beispielhaft eines Policy Types der Genauigkeit dar. Dieser besagt, dass das Topic, welches von einem Sensor beliefert wird, Werte bekommt, die eine Abweichung bis zu 0,5% beinhalten können. Das bedeutet, wenn einem Topic Consumer dieses Topics ein Wert von 30 erhält, muss er davon ausgehen, dass der originale Messwert zwischen 29,85 und 30,15 liegt.

Listing 4.6 Beispiel eines Policy Types der Abweichung

```
1 Deviation:
2     "deviation":  0.5
```

Guaranteed delivery Das Message Queuing Telemetry Transport (MQTT) Protokoll hat einen definierten QoS Standard [Gmb18]. Dieser bezieht sich auf die Nachrichtenübermittlung eines Topics. Es gibt drei Möglichkeiten mit welcher Garantie die Werte gesendet und empfangen werden. Die erste Form ist, dass ein neuer Sensorwert maximal einmal an alle Empfänger gesendet wird. Die zweite Form ist, dass der Wert mindestens einmal gesendet wird. Und die letzte Form ist, dass der Wert genau einmal bei den Empfängern ankommt. Da dies eine MQTT spezifische Angabe ist, kann der Policy Type nicht QoS heißen. Deshalb bekommt dieser Policy Type den Namen „Guaranteed delivery“ und besitzt ein Attribut. Dieses Attribut ist ein einfacher Zahlenwert zwischen Null und Zwei und folgt

damit der MQTT Definition. Listing 4.6 ist die Repräsentation der Attribute, die dieser Policy Type enthalten kann. In diesem Beispiel beträgt die MQTT QoS den Wert eins. Daraus folgt, dass das Topic die Werte **mindestens** einmal publiziert.

Listing 4.7 Beispiel eines Policy Types "Guaranteed delivery"

```
1 Deliver:
2     "mqtt_QoS":  1
```

Anhand der Beispiele sowie den Grundlagen von WS-Policy und der IoT können die Anforderungen an die Policys der TDLIoT festgelegt werden. Tabelle 4.1 gibt eine kurze Übersicht über die Anforderungen an die Policys. Die Vielfalt an möglichen nichtfunktionalen Anforderungen ist nicht limitiert. Gleichzeitig soll die Variation von Policy-Kombinationen nicht beschränkt sein. Somit können einem Topic beliebig viele Policys hinzugefügt werden die zueinander in konjunktiver oder disjunktiver Abhängigkeit stehen können. Daraus lassen sich verschiedene Anforderungen für die Umsetzung von Policys in der TDLIoT definieren. Die Struktur einer Policy muss die verschiedene Arten von Policys, die Anhand der unterschiedlichen Beispiele dargestellt wurden, abbilden können. Daraus entsteht (1) die Anforderung der Variantenvielfalt. Genau wie in der TDLIoT oder allgemein der IoT spielt Generizität eine wichtige Rolle. Da eine Policy aus verschiedenen Hintergründen wie Sicherheit, QoS oder eine unabhängige nichtfunktionale Anforderung stammen kann (2) muss das Modell einer Policy generisch gehalten sein. Damit soll gewährleistet sein, dass die Struktur jede Art von Policy abbilden kann und zusätzlich bei Änderungen einer Policy oder neuen Policy Types die Beschreibung niemals grundlegend neu definiert werden muss. Das verlangt, (3) dass eine Policy erweiterbar ist in ihrer Beschreibung und auch in der Menge an Policys die ein *Policy Provider* seiner Topic Beschreibung hinzufügen kann. Sonst könnte die TDLIoT im Bezug auf Policys an Grenzen kommen, sei dies die Anzahl an Policys oder auch die Relationen zwischen Policys. An den genannten Beispielen ist klar zu erkennen, dass eine Policy aus verschiedenen Angaben zusammengesetzt ist. Deshalb benötigt eine Policy der TDLIoT (4) eine Struktur mit komplexen Attributen. Sie kann nicht nur aus einem Schlüssel-Wert-Paar bestehen. Es müssen Verschachtelungen und dementsprechend Unterstrukturen existieren die jede Art von Policy repräsentieren können. Nicht jeder Topic Provider soll Policys definieren können wie es für seinen Anwendungsfall passend ist. Die TDLIoT fordert eine einheitlichen Beschreibung eines Topics. Dies gilt auch für die Policys, um das gewährleisten zu können, muss es (5) eine Instanz geben, die die Policys verwaltet und die bereits anerkannten Policys für die TDLIoT als Übersicht bereitstellt. Damit jeder Topic Provider aus der Sammlung an Policys die passende Policy für sein Topic hinzufügen kann. Dennoch muss es für jeden Anbieter eines Topics eine Möglichkeit geben, nichtfunktionale Anforderungen, SLA oder Definitionen von QoS als Policy zu beschreiben.

Im Katalog der TDLIoT wird nach Topics gesucht und gefiltert. Durch das Hinzufügen der Policy werden die Möglichkeiten, für den Topic Consumer, diesbezüglich sehr viel umfangreicher. Damit die Filterung und Suchfunktion auch für Policys der TDLIoT

Anforderung	Beschreibung
1	Variantenvielfalt
2	Generizität
3	Erweiterbarkeit
4	Komplexe Struktur
5	Übersicht der Policys
6	Such- und Filterfunktion
7	Lesbarkeit und Leichtgewichtigkeit

Tabelle 4.1: Anforderungen an die Policy der TDLIoT

funktioniert, muss als weitere Anforderung (6) die Such- und Filterfunktion angepasst werden. Wichtig hierbei ist, dass die Komplexität der Suchanfrage nicht an den Benutzer herangetragen wird, sondern die Logik im Hintergrund einfache Anfragen des Nutzers auf die verschachtelten Strukturen ausführt. Durch die Komplexität, Erweiterbarkeit und Generizität kann es zu einer schwierigen Übersicht und Lesbarkeit kommen. Deshalb ist eine weitere Anforderung (7) die Lesbarkeit und Leichtgewichtigkeit, welches mit der Anforderung Leichtgewichtigkeit der TDLIoT einhergeht, damit Topic Provider und Topic Consumer einen einfach Einstieg in die Thematik der Policys bekommen.

4.2 Policy4TDLIoT - Struktur und Aufbau

Aus den Anforderungen und den unterschiedlichen Policy Types lässt sich nun der Aufbau der Policys festlegen. Für den Aufbau und die Struktur sind die Anforderung (2) Generizität, (3) Erweiterbarkeit, (4) komplexe Attribute und (7) Lesbarkeit und Leichtgewichtigkeit von Relevanz. Auf dieser Basis wird das Konzept der Policy entworfen.

Es gibt keine festgelegte maximale Anzahl an Policys für ein Topic. Policys beschreiben einen Zustand oder eine Eigenschaft eines Topic. Dabei kann eine Policy sich in zwei verschiedene Kategorien einordnen. Entweder ist sie eine Beschreibung des Topics. Das kann eine Aussage über den Aufbau sein, wie der Zugriff drauf auszusehen hat oder welchen Limitierungen es ausgesetzt ist oder sie ist ein Beschreibung über die Nachricht die das Topic enthält. Bei ihr kann es Unterschiede in der Verschlüsselung geben, welche Genauigkeit die Werte haben oder in welcher Häufigkeit und Datenstruktur die Werte das Topic erreichen. In der Masse von möglichen Policys eines Topics kann es zu einer enormen Schwierigkeit werden die Übersicht zu behalten. Durch die Aufteilung in zwei Kategorien wird eine bessere Lesbarkeit sichergestellt. Trotz Aufteilung muss der Aufbau jeder Policy einheitlich sein. Das erhöht, genauso wie die zwei Kategorien, die Lesbarkeit. Hier entsteht jedoch ein Problem der Bezeichnungen von Policys eines Topics, da die Policy der TDLIoT die Gesamtheit aller Beschreibungen von einzelnen Policys definiert. Das bedeutet, dass übergeordnet der Begriff Policy das Element in der TDLIoT beschreibt,aber

jeder Policy Type untergeordnet in die Kategorie Policy fällt. In der WS-Policy [WCL+05] werden Policy Assertions aufgestellt, also Aussagen über Policy Beschreibungen. Das gleiche Konzept kann auf die Policys für die TDLIoT umgesetzt werden. In Abbildung 4.1 ist dieses Modell der Policy dargestellt. Es ist ein ER-Modell nach Chen [Che76] welches die Verbindung zwischen den einzelnen Entitäten aufzeigt. Die beiden Kategorien „topic“ und „message“, sind jeweils optional, sobald eine Policy Assertion der Policy angefügt werden soll, muss sie der richtigen Kategorie zugeordnet werden. Daraus folgt, dass es Policys ohne die Topic Kategorie geben kann, aber nur wenn es keine Policy Assertion für das Topic gibt. Das erleichtert die Übersicht für die gesamte Policy, denn so werden nur vorhandene und relevante Informationen in der TDLIoT beschrieben.

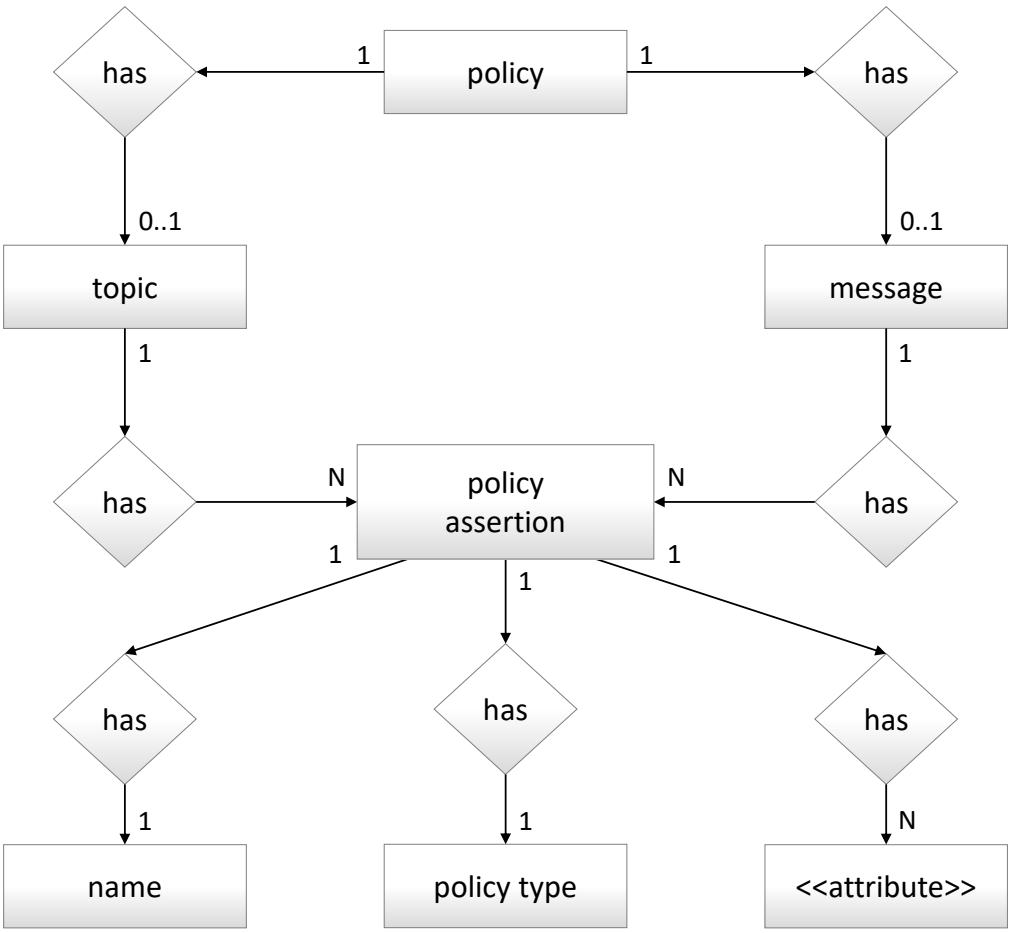

Abbildung 4.1: Modell der Policy

Eine Policy Assertion ist die Instanz eines Policy Types und bildet die in der Analyse genannten Beispiele ab. Deshalb spielen die Anforderungen Generizität und Erweiterbarkeit für die Policy Assertion eine wichtige Rolle. Für das Verständnis und die Lesbarkeit muss jede Policy Assertion dieselbe Struktur aufweisen. Jede Policy Assertion besteht aus mehreren Angaben um beispielsweise eine, für das Topic, nichtfunktionale Anforderung

zu beschreiben. Jede Assertion benötigt verschiedene Attribute zur Beschreibung. Aber es müssen auch Gemeinsamkeiten zwischen den Policy Assertions gefunden werden können. Für die Topic Provider und Consumer ist es wichtig, dass es für dieselbe Eigenschaft oder Verhalten immer dieselbe Policy Assertion ist, welche dies beschreibt. Außerdem verwendet jedes Topic bei der Definition der Kosten die gleiche Policy Assertion. Daraus folgt, dass eine Policy Assertion ein Attribute benötigt, das definiert zu welchem Policy Type diese Assertion gehört. Dieses Attribut ist dann Indikator für den Aufbau dieser Policy Assertion und lässt dadurch unterschiedliche Varianten einer Assertion zu. Somit besteht beispielsweise die Kosten Policy Assertion immer aus den drei gleichen Attributen aber die Verfügbarkeit Assertion aus zwei anderen Attributen. Dennoch bleibt die Beschreibung derselben Policy Types identisch und ermöglicht die Wiedererkennbarkeit. Der Policy Type selbst gibt keine genaueren Auskünfte über dessen Angaben. Zusätzlich wird ein Attribut zur Beschreibung der Policy Assertion benötigt. Dieses Attribute heißt „name“ und hilft dem Topic Consumer zu verstehen, für welchen Zweck dieser Policy Type dem Topic hinzugefügt wurde. Gleichzeitig kann eine Abgrenzung zu einem anderen Topic, welches dieselben Daten liefert, gemacht werden. Beispielsweise kann es zwei Topics vom selben Anbieter geben, die dieselben Daten in verschiedenen Intervallen liefern und dadurch die Kosten für das Anmelden unterschiedlich sind. Mit dem beschreibenden Attribute „name“ kann somit eine Abgerenzung zwischen diesen beiden Topics ermöglichen in dem der Name der teureren Policy „high cost“ und der niedrigeren „low cost“ heißt. Diese beiden Attribute, „name“ und „policy_type“, müssen in jeder Policy Assertion existieren. Jeder Policy Type beinhaltet eigene Angabe für seine Beschreibung. Diese werden durch zusätzliche Attribute repräsentiert. Damit die Policy Assertions erweiterbar bleiben gibt es keine Begrenzung für die Anzahl an zusätzlichen Attributen. So lassen sich auch komplexe Policy Types abbilden. Weitere Restriktionen der Policy Assertions gibt es nicht. Alle Policy Assertions, seien sie in der Kategorie Topic und in Nachricht, sind konjunktiv verknüpft. Das bedeutet es gibt in einer Beschreibung eines Topics keine Variation aus verschiedenen Policy Assertions, sondern alle Assertions müssen gelten. Dadurch bleibt die Beschreibung kürzer und lesbarer. Gleichzeitig wird es leichter die Policy zu erweitern. Die Variationsmöglichkeit ist somit nicht in einer Beschreibung gegeben. Sie kann dennoch in der gesamten TDLIoT vorhanden sein, denn es können mehrere Topics mit unterschiedlichen Beschreibungen erstellt werden. Dadurch lassen sich alle Variationen an Policy Assertion Verbindungen definieren. Die Auswirkungen werden in Abschnitt 4.3 genauer erläutert.

Listing 4.8 Beispielhafte Policy Assertion

```
{
    "policy_type": "Pricing",
    "name":        "small package",
    "cost_period": "month",
    "cost":        1.19,
    "currency":    "euro"
}
```

Listing 4.8 zeigt beispielhaft eine Policy Assertion welche die Kosten für das Topic beschreibt. Der Policy Type beschreibt um welche Art von Policy Assertion es sich handelt und definiert welche zusätzlichen Attribute außer dem Namen vorhanden sein müssen. In dem Fall gehört zur Policy Assertion „Pricing“ der Zeitraum, in dem die Kosten anfallen, die Höhe des Preises und um welche Währung es sich handelt. Die Höhe der Kosten werden als Dezimalzahl angegeben. Die anderen beiden Angaben sind textueller Form und werden mit Schlüsselwörtern definiert damit alle Topic Provider, welche die Policy Assertion mit Werten füllen, diese einheitlich beschreiben. Damit lassen sich alle möglichen Währungen definieren aber sie werden von allen gleich beschrieben. Analog sind alle Arten von Zeiträume möglich aber die Beschreibung für jedes Topic sieht immer einheitlich aus. Das Attribute „name“ kann frei beschrieben werden. Bei einer Benennung kann der Topic Provider ein kleines Paket mit niedrigeren Kosten, dafür mit mehr Einschränkungen, definieren und ein großes Paket, was mehr kostet aber dementsprechend mehr Qualität in Genauigkeit oder Intervall liefert. So kann jeder Topic Provider flexibel seine Topics beschreiben.

Listing 4.9 Beispielhafte Policy in der TDLIoT

```
"policy": {
    "topic": [
        {
            "policy_type": "Pricing",
            "name":        "small package",
            "cost_period": "month",
            "cost":        1.19,
            "currency":    "euro"
        }
    ],
    "message": [
        {
            "policy_type": "Accuracy",
            "name":        "Accuracy",
            "accuracy":    0.001
        }
    ]
}
```

Listing 4.9 zeigt ein Beispiel einer Policy der TDLIoT. Alle Policy Assertions gliedern sich den Kategorien „topic“ und „message“ unter. In diesem Fall sind in beiden nur jeweils eine Assertion vorhanden, im allgemeinen ist die Anzahl der Policy Assertions nicht beschränkt. Da der Policy Type eindeutig ist, kann jede Policy Assertion nur einmal in einer Policy vorkommen. Das schränkt die Möglichkeiten der Beschreibung für ein einzelnes Topic ein, kann aber durch mehrere Topic Beschreibungen des selben Sensors gelöst werden. Es können Konflikte im Bezug auf Attribute unterschiedlicher Policy Types entstehen. Wird beispielsweise in zwei Policy Types das Attribut „cost“ verwendet, bei einem ist es als Ganzzahl und beim anderen als Dezimalzahl definiert, ist bei einer Instanziierung unklar inwiefern dieses Attribut beschrieben werden soll. Für dieses Problem gibt es

zwei Lösungsvarianten. Es kann bei der Instanziierung angegeben werden für welchen Policy Type das Attribut gilt oder es muss verhindert werden, dass zwei Attribute den gleichen Namen tragen können. Die ersten Variante benötigt eine klare Beschreibung für welchen Policy Type die Policy Assertion erstellt wird. Bei der zweiten Variante muss bei der Erschaffung eines neuen Policy Types eine Überprüfung angestoßen werden, die alle anderen Policy Types auf eine Überschneidung überprüft und gegebenfalls die Erstellung verhindert oder dem Attribut einen neuen Namen gibt. In XML sind für diesen Fall *Namespaces* definiert, die angeben, aus welcher Definition die Beschreibung abgeleitet wird. Für Datenformate wie JSON und YAML, die in der TDLIoT verwendet werden, sind im Bereich der *Namespaces* keine Beschränkungen vorhanden, bieten dafür aber keine Unterstützung zur Unterscheidung.

Die Integration der Policy in die TDLIoT beeinflusst die Gesamtstruktur, die in [FHB+18] dargestellt wird. Abbildung 4.2 ist die erweiterte Struktur dargestellt, diese besteht aus den alten und neuen Rollen sowie Komponenten. Komponenten speichern Datensätze und stellen diese zur Verfügung, Rollen dagegen nutzen und bearbeiten diese Daten. Alle Elemente die eine schmale Umrandung besitzen gehören zur TDLIoT. Das ist der *Topic Catalog*, er enthält alle Topics die durch die TDLIoT bisher beschrieben wurden. Der Topic Consumer hat die Möglichkeit durch die Such- und Filterfunktionalität den Katalog nach Topics zu durchsuchen. Wenn er das gewünschte Topic gefunden hat sind in der Beschreibung alle Informationen zum Aufbau einer Verbindung enthalten. Somit kann er das Topic für seine Zwecke verwenden. Ein Topic Provider kann dem Topic Catalog eine neue Beschreibung eines Topics hinzufügen.

Neu hinzugekommene Komponenten haben in der Abbildungen eine stärkere Umrandung. Mit Hilfe der *Policy Collection* kann ein Topic Provider die passenden Policys seines Topics finden. Sie ist die Sammlung aller Policy Types die bereits als valide Beschreibung überprüft und verifiziert wurden. Hier gibt es eine Trennung zwischen verifizierten Policy Types und nicht verifizierten damit die Qualität eines Policy Types für den Topic Provider sichergestellt ist. Aber auch damit keine Policy zweimal auf unterschiedliche Art und Weise beschrieben wird. Genauer wird die Policy Collection im folgenden Abschnitt 4.2.1 beschrieben. Als Quelle der Policy Collection steht das *Policy Repository*. In diesem Repository sind alle Policy Types gespeichert die bisher definiert wurden. Zwei Rollen können damit interagieren. Das ist zum einen der Policy Provider dieser erstellt nach seinem Bedürfnissen und Anwendungsfällen Policy Types. Der Policy Expert ist die kontrollierende Rolle, welche vom Policy Provider erstellte Policy Types überprüft und wenn diese eine valide Beschreibung darstellen als verifiziert markiert. Als Alleinstellungsmerkmal in der neuen Struktur der TDLIoT gilt, dass die Rolle des Policy Expert nur von einem Menschen übernommen werden kann, der das Fachwissen der Policys besitzt. Das kann sich je nach Policy vom rechtlichen Wissen bis hin zum technischen Wissen erstrecken. Die verifizierten Policy Types werden von der Policy Collection erfasst und in ihrer Sammlung dargestellt. Die Komponente Policy Repository wird in Abschnitt 4.2.2 mit den beiden Rollen Policy Expert und Policy Provider näher erläutert.

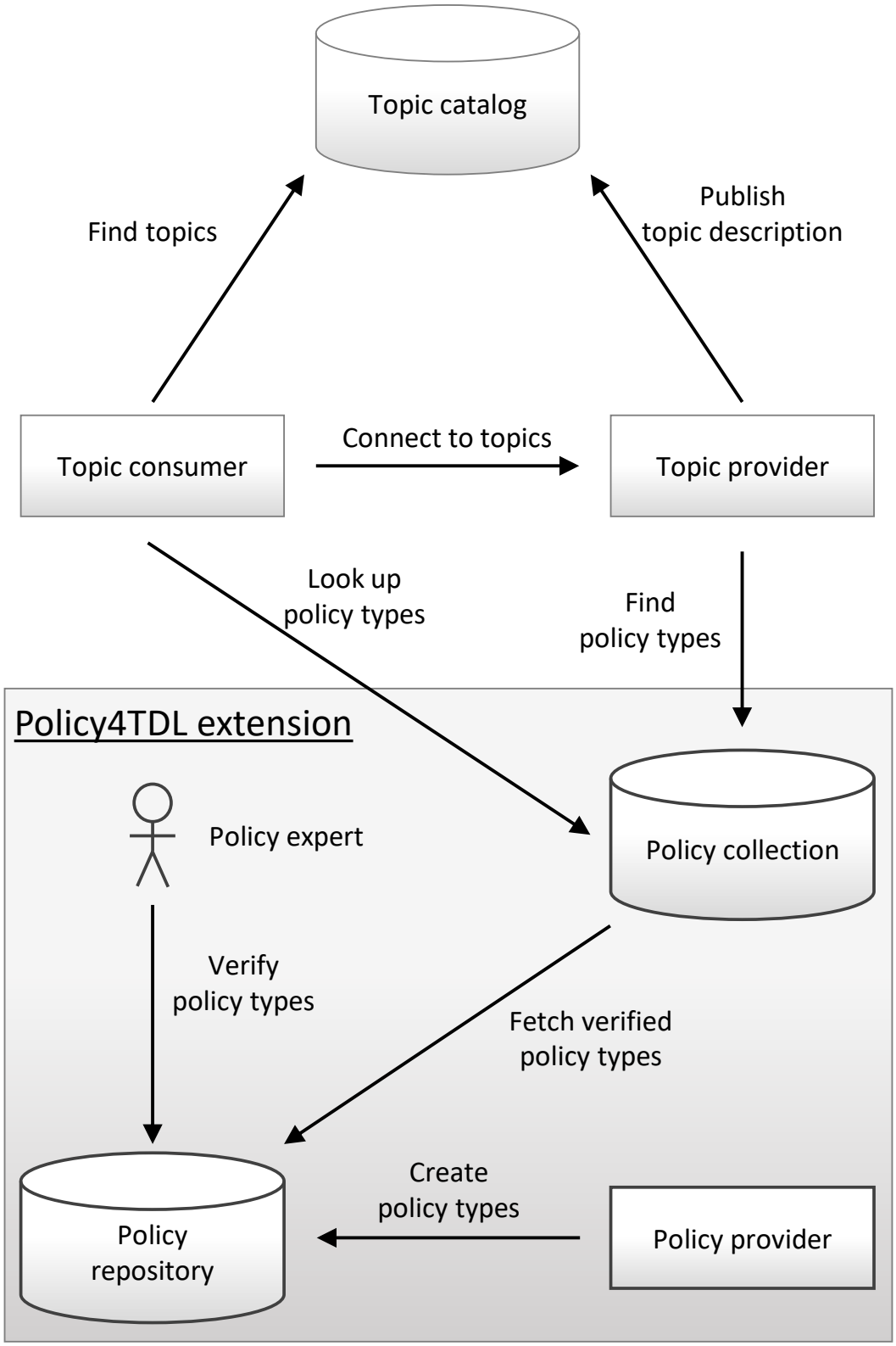

Abbildung 4.2: Übersicht der Verhältnisse zwischen Rollen und Komponenten der TDLIoT nach Einbeziehung der Policys

4.2.1 Policy Collection

Um dem Topic Provider die Möglichkeit zu bieten bereits existierende Policy Types zu verwenden, muss es eine Übersicht dafür geben. Als Anforderung (5) Übersicht der Policys ist dies ein wichtiger Aspekt dieser Arbeit. Eine Übersicht über alle Topics soll dem Topic Provider helfen, das Topic zu finden welches seinen Anwendungsfall beschreibt und dem Topic Consumer als Hilfestellung dienen um zu verstehen was die Beschreibung einer Policy für das Topic bedeutet. Alle Informationen zu einer Policy soll die Übersicht auf einen Blick bereithalten. Das Grundprinzip dieser Übersicht orientiert sich an [Hir18] und [FBFL14]. Dort werden komplexe Strukturen in einer Sammlung dargeboten, die die wichtigsten Inhalte jeder Elements darstellt und eine Navigation zur Findung von neuen Elementen zu bieten. Abbildung 4.3 stellt das Konzept Policy Collection dar. Jeder Policy Type wird als Auswahl bereitgestellt. Beim Öffnen einer dieser Elemente wird eine genauere Beschreibung der Policy sichtbar. Zur detaillierten Beschreibung gehört eine Erklärung der Policy für welchen Zweck die Policy Types zu verwenden sind. Darunter werden alle Attribute aufgelistet mit ihrem Datentyp sowie der einzelnen Beschreibung, um zu zeigen für welche Angabe dieses Attribute steht. In der letzten Spalte der Tabelle kann der Ersteller eines Topics seine gewünschten Werte eintragen und durch eine Aktion, wie beispielsweise das Drücken eines Buttons die erstellte Policy dem Topic hinzufügen. Zum Schluss gehört ein kurzes Beispiel zur detaillierten Beschreibung. So kann jeder Topic Provider nachvollziehen wie er die Policy zu definieren hat und welchen Fall dieser Policy Type abdeckt. Gleichzeitig ist es das Nachschlagewerk für jeden Topic Consumer, um zu verstehen, was die einzelnen Attribute eines Policy Types bedeuten.

4.2.2 Policy Repository

Die Informationen, welche die Policy Collection über die einzelnen Policy Types darstellt, sollten kompakt an einer zentralen Stelle gespeichert sein. Dieser Speicherort muss verschiedenen Anforderungen genügen. Er sollte zulassen, dass auf einfache Weise neue Policy Types definiert werden können. Zudem sollte der Zugriff darauf unabhängig vom TDL-Katalog möglich sein, um flexibel damit zu interagieren. Die letzte Anforderung ist eine Zugriffsverwaltung, da es für jeden Policy Provider erlaubt sein soll Policy Types zu erstellen, aber nur für ausgewählte Personen, den Policy Expert, möglich sein soll Policy Types für die Policy Collection freizugeben. Nur so kann garantiert werden, dass für jede Problemstellung nur ein spezifischer Policy Type existiert.

Für diese Anforderungen bietet sich ein Repository in einer cloud-basierten Umgebung an. Die Zugriffsverwaltung ist durch verschiedene Zweige der Versionierung umsetzbar. Auf den einen Pfad kann jeder zugreifen und ihn erweitern. Der andere Pfad beinhaltet nur die Policy Types die bereits überprüft und für die Sammlung der Policys, die Policy Collection, zugelassen wurden. Abbildung 4.2 ist die Rolle des Policy Provider, in Zusammenhang mit dem Policy Repository, dargestellt. Ein Policy Provider kann unabhängig von Topics und Topic Providern Policy Types definieren. Ihm stehen dafür alle Freiheiten neue Policy

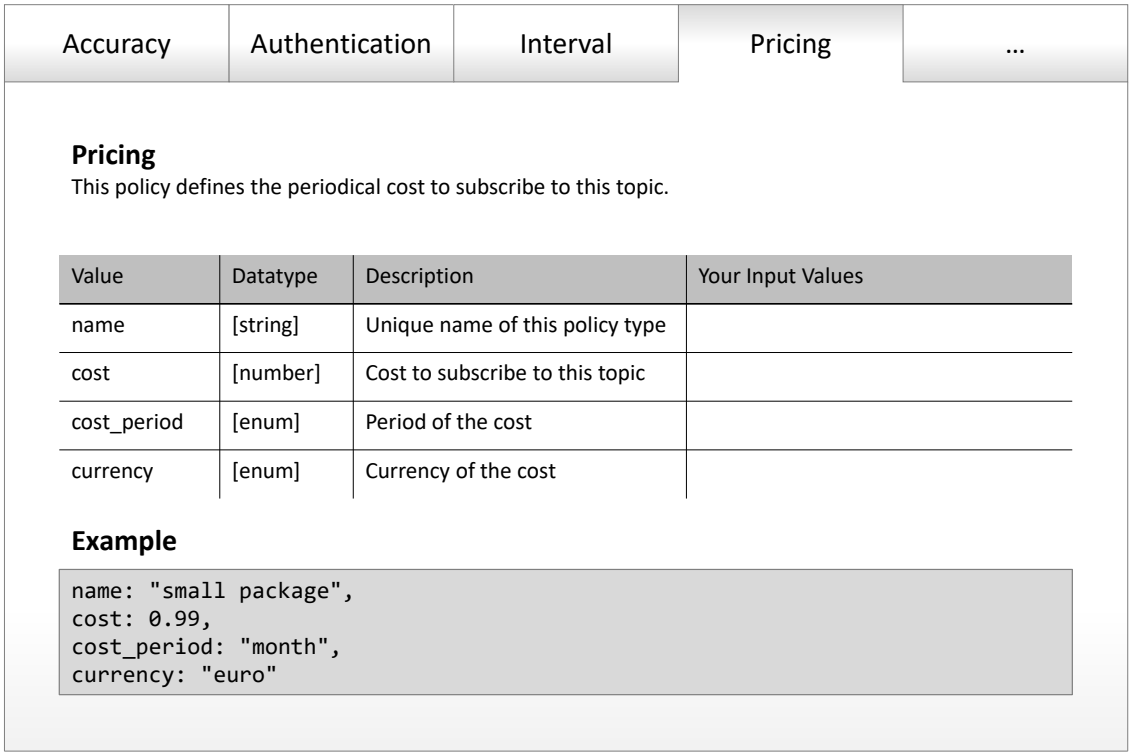

Abbildung 4.3: Graphisches Konzept der Policy Collection

Types zu beschreiben oder bisherige für seinen Anwendungsfall anzupassen zur Verfügung. Das bedeutet nicht, dass eine Person nicht mehrere Rollen übernehmen kann. Gerade die Rollen des Topic Providers und Policy Providers können sich gegebenenfalls überschneiden. Wenn Privatpersonen oder kleinere Institutionen ein Topic bereitstellen muss sich eine Person mit den Policys dafür auseinandersetzen. Das macht in diesem Fall in der Regel eine Person mit den nötigen Kenntnissen zum Topic. In einer größeren Institution wird gegebenenfalls jede Rolle einzeln besetzt, da die Policys in ihrem Fall einen größeren rechtlichen Hintergrund haben kann, gerade im Bezug auf SLAs. Deshalb würde hier der Topic Provider ein Mitarbeiter mit dem nötigen technischen Wissen sein und der Policy Provider jemand mit einem rechtlichen Hintergrund. Die Aufgabe der Überprüfung von neuen oder geänderten Policy Types wird der Rolle des Policy Expert zugewiesen. Jede Person mit dieser Rolle hat vollen Zugriff auf das Policy Repository und ist wichtigster Einfluss für die Qualität der Policys der TDLIoT. Die Aufgaben des Policy Expert sind die neu erstellten Policy Types auf Duplikate und Qualität der Beschreibung zu überprüfen. Deckt noch kein anderer Policy Type den beschrieben Zweck ab und ist die Erklärung des Policy Types eindeutig und leicht verständlich, kann er es als verifiziert markieren. Die Policy Collection wird dann diesen Policy Types in ihre Sammlung aufnehmen.

Jeder Policy Type wird mittels einer einzelnen Datei im Repository beschrieben. Die Struktur der Beschreibung kann durch eine vereinfache Auszeichnungssprache wie JSON oder YAML erfolgen. Dies erlaubt, durch den definierten Aufbau der Sprache, eine klare

Listing 4.10 Beispiel der Beschreibung eines Policy Types im Repository

```
1
2  policy_type: accuracy
3  description: Describes the accuracy of all published values of the topic
4  policy_category: message
5  input:
6    - value: name
7      datatype: string
8      description: Unique name of this policy
9    - value: accuracy
10     datatype: number
11     description: Accuracy of the sensor publishing values to the topic
12 example:
13   name: accuracy
14   accuracy: 0.001
```

und detaillierte Beschreibung eines Policy Types. In Abschnitt 4.2 wird beschrieben wie eine Policy Assertion, die einen Policy Types abbildet, aufgebaut ist. In der Policy Collection müssen alle Attribute einer Assertion dargestellt sein. Zudem werden Erklärungen und Beispiele für das Verständnis einer Policy Assertion benötigt. In dem Beispiel des Listing 4.10 wird exemplarisch gezeigt aus welchen Angaben die Beschreibung besteht. Jeder Policy Type muss einen eindeutigen Namen besitzen und als Attribut in der Datei vorhanden sein. Die allgemeine Beschreibung des Policy Types soll dem Topic Provider in einer kurzen Erklärung verdeutlichen, für welchen Zweck dieser Policy Type verwendet werden soll. Die Kategorie ist die Eingliederung in die Policy für die TDLIoT. Es gibt zwei Kategorien, die ein Ersteller eines Policy Types verwenden kann, das sind „topic“ und „message“. Danach werden alle Attribute, bis auf den bereits definierten Policy Types, als Liste beschrieben. Jede Beschreibung eines Attributs besteht aus drei Angaben, welche die ersten drei Spalten der Tabelle in der Policy Collection darstellen. Alle Angaben die hier definiert sind beschreiben was ein Nutzer, also Topic Provider, für dieses Attribut angeben kann. Das ist zum einen der Wert, dieser enthält einen einzelnen Begriff der als Schlüsselwort für die Beschreibung des Attributs steht. Als nächstes wird der Datentyp dieses Attributs definiert. Der Datentyp ist auf Text, ganze Zahlen, Dezimalzahlen, boolesche Werte und fest definierte Schlüsselwörter begrenzt. Die letzte Angabe ist die Beschreibung des Attributs, die dem Topic Provider erklären soll welchen Einfluss dieses Attribut auf den Policy Types hat. Zum Schluss muss ein Beispiel für den Policy Types definiert werden. Dieser ist ein wichtiger Bestandteil der Policy Collection, um dem Topic Provider einen leichten Einstieg in diesen Policy Types zu geben.

4.3 Erweiterungen und Integration in die TDLIoT

Bisher wurde der Aufbau einer Policy beschrieben und mit welchen Rollen und Komponenten die Struktur der TDLIoT erweitert werden muss. Nicht nur um Policys als Topic Provider sinnvoll zu nutzen sondern auch damit die Aussagekraft und Qualität einer Policy sichergestellt ist. In diesem Abschnitt wird die Integration der Policy in die TDLIoT dargestellt. Zusätzlich werden die Änderungen die an dem bisherigen Topic Catalog für das neue Modell der Policy vorgenommen werden müssen konzeptionell erklärt.

Abbildung 4.1 ist das Modell der Policy aufgezeichnet. Für die Integration muss die Policy in das Modell der TDLIoT eingegliedert werden. Da in den Anforderungen der TDLIoT Erweiterbarkeit enthalten ist und der Aufbau dies widerspiegelt, muss die Policy als weiteres Unterelement hinzugefügt werden. Die Abbildung 4.4 zeigt die Integration als ER-Modell. Ein Topic kann optional Policy enthalten, genauso wie das Element *unit* der TDLIoT. Durch die zwei Unterkategorien der Policy und den möglichen Policy Assertions ist die Policy das komplexeste Attribute der Topic Beschreibung und ist in der Abbildung grau hinterlegt.

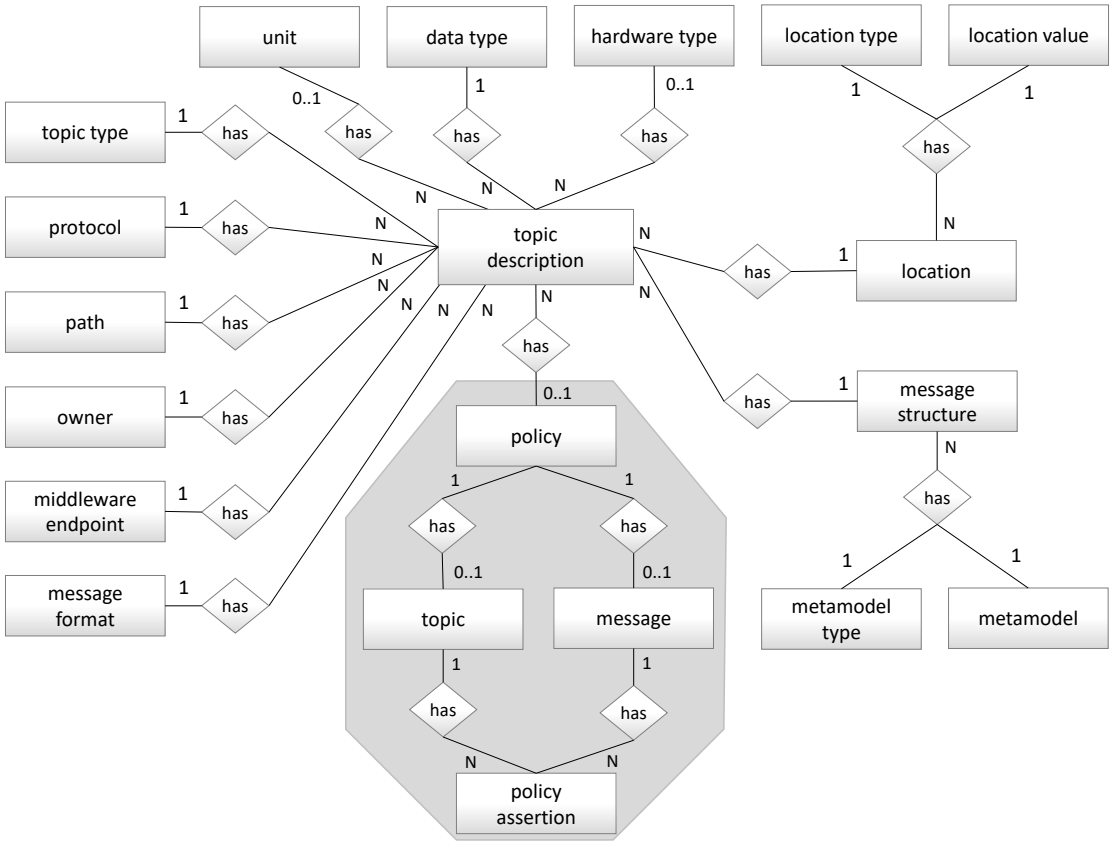

Abbildung 4.4: Modell der TDLIoT und der integrierten Policy

Die Anforderung (1) Variantenvielfalt für die Policy, ist in der Konzeption auf ein Topic beschränkt. Das kommt daher, dass die Relation aller Policy Types, einer Policy der TDLIoT, untereinander konjunktiv verknüpft sind. Durch dieses Verhältnis lassen sich nicht alle Anwendungsfälle abbilden. Falls die Kosten für die Verbindung zu einem Topic sich verändern, je nachdem in welchem Intervall neue Daten erhalten werden sollen, kann das in der aktuellen Struktur der TDLIoT nicht umgesetzt werden. Nach Franco da Silva et al. [FHB+18], ist jeder Sensor mit einem einzigen Topic verbunden. Dieses Modell ist in Abbildung 4.5 dargestellt.

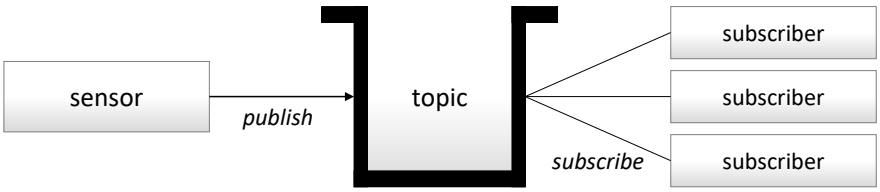

Abbildung 4.5: Sensor-Topic-Modell der TDLIoT

Bisher gab es keine Relevanz für eine Änderung dieser Struktur. Doch um der Anforderung der Variantenvielfalt gerecht zu werden, muss hier eine Anpassung der Struktur der TDLIoT durchgeführt werden. Für diese Anpassung gibt es zwei Möglichkeiten. Zum einen könnte das Prinzip, dass ein Sensor genau ein Topic besitzt beibehalten werden. Dafür müsste eine *Middleware* zwischen jedem *Subscriber* und Topic geschalten werden. Dadurch wäre die Komplexität der Variantenvielfalt der Middleware zugewiesen. Sie wäre dafür zuständig, dass je nach Vertrag zwischen Topic Provider und Topic Consumer die richtigen Daten übertragen werden. Die zweite Variante ist die Struktur zwischen Topic und Sensor zu verändern und jedem Sensor die Möglichkeit zu bieten mehrere Topics mit Daten zu beliefern. Dann wäre es möglich, dass unterschiedliche Beschreibungen von Topics in der TDLIoT auf den selben Sensor zugreifen. Nachteil von der zweiten Variante im Gegensatz zur ersten ist, dass die Anzahl an Topics die in der TDLIoT beschrieben sind wesentlich höher ist. Der Vorteil dagegen ist, dass die Vertragsverhandlungen zwischen Anbieter und Nutzer eines Topics sich auf ein Minimum reduziert, da alle möglichen Angebote der Daten eines Topics sich auf die beschränkt, die in der TDLIoT vorhanden sind. Zudem wird keine dritte Schnittstelle, also keine Middleware, die als Verhandlungskomponente fungiert benötigt. Damit bleibt die Struktur für die Subscriber erhalten und leichtgewichtig. Der Nachteil, der erhöhten Anzahl von Topics, kann durch eine flexible und strukturierte Such- und Filterfunktion im Katalog behoben werden. Deshalb wird die zweite Möglichkeit für die TDLIoT verwendet. Abbildung 4.6 ist diese dargestellt. Somit wird die Aussage, dass jeder Sensor ein einziges Topic bedient, abgeändert und definiert, dass jeder Sensor **mindestens** ein Topic besitzt. Die Anzahl an Subscribern pro Topic ist im neuen sowie im alten Modell beliebig erweiterbar und nicht auf die Menge beschränkt, wie in den Abbildungen zu sehen ist.

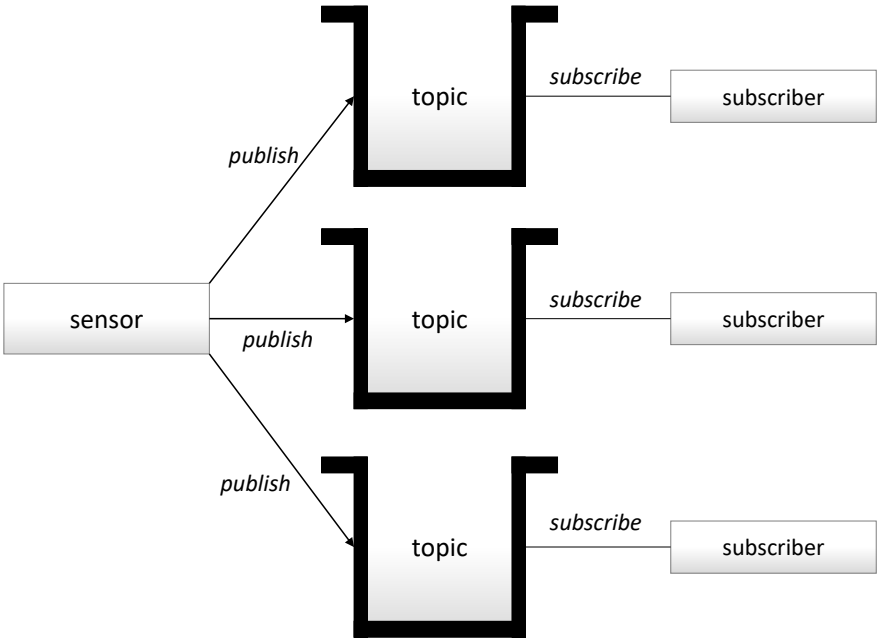

Abbildung 4.6: Erweitertes Sensor-Topic-Modell der TDLIoT

Durch die Integration der Policy und die Änderung des Aufbaus zwischen Topic und Sensor muss die Such- und Filterfunktionalität für Topics in der TDLIoT angepasst werden. Gleichzeitig ist es eine Anforderung an die TDLIoT nach Einbeziehung der Policy, da die Komplexität zur Filterung deutlich erhöht werden muss. Wichtig ist dafür, dass die Filterung für den Suchenden, trotz erhöhter Komplexität, einfach zu bedienen ist. Das bedeutet, dass bei einem Szenario in dem ein Topic Consumer ein Topic sucht, welches Temperaturdaten jede Stunde für unter einem Euro pro Monat anbietet, der Consumer nur diese drei Suchkriterien eingeben muss. Er soll zudem nicht die Struktur einer Policy verstehen müssen. Darum wird die Aufgabe sein, eine einfach Eingabe mit Schlüssel-Wert-Paaren in eine Suchanfrage umzuwandeln, die die unterschiedlichen Policys eines Topics nach den Schlüsseln und Schlüsselwerten durchsucht. So wird dem Suchenden eine hohe Benutzbarkeit gewährt.

4.4 Policy-Verifikation

Jeder Topic Provider wählt für die Beschreibung seines Topics die Policys aus, welche für seinen Anwendungsfall passend sind. Das führt dazu, dass die Möglichkeit des Betrugs besteht.

Wenn ein Anbieter mittels einer Policy angibt, das sein Topic Werte im 100 Millisekunden Intervall liefert. Dies aber nicht der Wahrheit, egal ob unabsichtlich oder absichtlich, kann für jeden Topic Consumer, der davon ausgeht, dass die Angaben richtig sind, Schwierigkeiten für sein System entstehen, die dazu führen können, dass der Topic Consumer die Vereinbarungen mit seinen Kunden nicht einhalten kann. Um das zu vermeiden soll die TDLIoT eine Verifikation der Policys bereitstellen. Diese erlaubt jedem Topic Consumer die Übersicht ob eine Policy eines Topics eingehalten wird oder nicht. Dazu gehört auch das Aufzeigen ob ein Topic mit dem angegeben Pfad erreichbar sein kann. Solche Unstimmigkeiten der Angaben zu einem Topic müssen nicht mit Absicht definiert worden sein. Gerade bei Änderungen bestehender Topics bezüglich ihrer Domäne oder Policys können Fehlangaben unabsichtlich auftreten. In machen Fällen kann bei Policys keine Verifizierung durchgeführt werden. Ein weiter Aspekt ist, wenn eine Policy besagt, dass der Sensor, der das Topic beliefert, eine Abweichung von 0,5% beinhalten kann, gibt es keine Möglichkeit zu überprüfen ob manche Werte eine größere Abweichung vom Originalwert haben als definiert. Deshalb muss es verschiedene Zustände der Verifizierung geben. Als Übersicht sind diese in Tabelle 4.2 dargestellt. Die ersten beiden sind Aussagen über Policys die verifiziert werden können und die Zustände liefern das Ergebnis dieser Überprüfung. Der dritte Zustand zeigt, dass die Überprüfung noch vollzogen wird, und noch keine Aussage über ein positives oder negatives Ergebnis getroffen werden kann. Aus diesem Zustand muss ein erfolgreiches oder fehlgeschlagenes Ergebnis folgen. Der letzte Verifizierungszustand ist für alle Policys vorgesehen, die nicht überprüft werden können. Die Informationen der Zustände einer Policy sollten im Katalog gespeichert werden, da dort alle Topics mit ihren Policys enthalten sind und die Zustände abhängig von den Angaben der Policys sind.

Zustand	Beschreibung
erfolgreich	Nach der Verifizierung einer Policy und einem erfolgreichen Ergebnis
fehlgeschlagen	Nach der Verifizierung einer Policy und einem fehlgeschlagenen Ergebnis
in Bearbeitung	Während der Verifizierung einer Policy
unbekannt	Eine Policy die nicht verifiziert werden kann

Tabelle 4.2: Zustände der Verifizierung einer Policy

4.5 Anforderungsevaluation

Die Konzeption beinhaltet die Erfüllung der gestellten Anforderungen. Als Indikator für die Evaluation der Anforderungen muss die Frage gestellt werden, ob die Anforderung durch die Analyse für die Konzeption erfüllt wurden. Gleichzeitig soll aufgezeigt werden

an welchen Stellen Potenzial für Erweiterungen sind und welche Anforderungen durch Limitationen beschränkt sind. Tabelle 4.1 sind die Anforderungen, die an die Policy gestellt wurden abgebildet.

Die erste Anforderung ist die Variantenvielfalt, diese bezieht sich auf die verschiedenen Arten von Policys. Dazu gehört auch die Möglichkeit einem Topic eine beliebige Kombination aus Policy Types hinzuzufügen. Hier gibt es in der Konzeption eine Beschränkung, da alle Policy Types eines Topics in konjunktiver Relation zueinander stehen. Diese Einschränkung wurde aber durch die Anpassung des Sensor-Topic-Verhältnisses aufgebrochen. Da mit der Erweiterung jeder Sensor mehrere Topics beliefern kann und diese mit unterschiedlichen Policy Types definiert sein können. So lässt sich die Variantenvielfalt wiederherstellen. Die Anforderung der verschieden Arten von Policys bezieht sich zusätzlich auf andere Anforderungen, wie die komplexe Struktur oder Lesbarkeit und werden in den folgenden Evaluationen beschrieben.

Die Anforderung der Generizität ist im gleichen Maße für die Policy relevant wie in der allgemeinen TDLIoT. Die Anforderungsanalyse hat gezeigt, dass eine Policy aus sehr unterschiedlichen Bereichen stammen kann. Sei es eine protokollspezifische wie bei MQTT, wertabhängige wie des Temperatursensors oder eine Beschreibung, die sich um den vertraglichen Rahmen handeln, wie die Kosten für das Anmelden bei einem Topic. Diese Unabhängigkeit der Domain spielt in der gesamten IoT eine wichtige Rolle. Der Aufbau einer Policy lässt ohne Einschränkung diese Generizität zu.

In Anbetracht, dass Policys sehr unterschiedlich sein können und die Möglichkeit besteht, dass sie sich im Laufe der Jahre verändern, müssen Policys erweiterbar sein. Dies erlaubt die Modellierung einer Policy da beliebig viele Attribute einem Policy Types hinzugefügt und der Policy beliebig viele Policy Types angefügt werden können. Bei einer Änderung eines existierenden Policy Types kann im Policy Repository darauf zugegriffen werden und diese wird nach der Verifikation in die Policy Collection eingebettet. Hier spielt der Policy Expert eine wichtige Rolle im Bezug auf Konflikte. Er ist die einzige Instanz, die überprüft ob durch die Änderung Konflikte zu der bisherigen Definition des Policy Types entstehen oder ob Konflikte und Überschneidungen zu anderen Policy Types entstehen. Durch diese Kontrollstruktur wurde das Policy Repository erstellt, welches als Werkzeug für den Policy Expert dient und als Unabhängiger Anlaufpunkt für jeden Policy Provider fungiert. Somit ist die Anforderung in ihren verschieden Aspekten erfüllt.

Die TDLIoT besitzt ohne die Policys hauptsächlich einfache Attribute. Sie bestehen aus einem Namen des Attributs und dessen Wert, also ein Schlüssel-Wert-Paar. Mit der Anforderung an eine komplexe Struktur, um die verschiedenen Arten von Policys in einem Policy Element darstellen zu können, wird die Idee aus WS-Policy [WCL+05] von komplexen Strukturen verwendet. Das Modell der Policy in Abbildung 4.1 zeigt die Verschachtelung der Policy Types und dessen Zusammenhang zur gesamten Policy. Abbildung 4.4 zeigt wie die sich die Policy in das Topic eingliedert. Die Struktur zeigt die Komplexität der Policy auf und erfüllt diese Anforderung.

Jeder Topic Provider soll für seine Beschreibung eines Topics die Policys verwenden, die bereits von anderen Topic Providern benutzt wurden. Damit es für einen Zweck nur eine definierte Policy existiert. Deshalb wurde die Anforderung einer Sammlung von Policys gestellt, die ein Topic Provider für seine Beschreibung nutzen kann. Dafür ist die Policy Collection konzipiert, um dem Nutzer einen schnellen und einfachen Überblick zu beschaffen. Dazu gehört eine detaillierte Darstellung eines Policy Types mit allen Attributen, einer kurzen erklärenden Beschreibung und einem Beispiel. Mit dieser Policy Collection ist die Anforderung einer Übersicht der Policys erfüllt.

Die Anforderung der Such- und Filterfunktionalität bezieht sich auf eine Verbesserung der bisherigen Funktionalität. Da durch die Verschachtelung der Policy die Möglichkeit bestehen soll, nach tiefer liegenden Attributen der Policy Types zu filtern und zu suchen. Zum anderen gilt es Bereichsabfragen zu definieren. Das bedeutet, dass ein Topic welches Daten in einer bestimmten Preisspanne anbietet, gefunden werden kann. Deshalb bezieht sich die Anforderung zusätzlich auf die Erweiterung der Vergleichsoperatoren. Über eine einfache Gleichsetzung soll eine größer, kleiner und ungleich Operation sowie eine Ober- und Untergrenze als Such- und Filteroperation integriert werden. Diese Erweiterung bezieht sich auf den Katalog der TDLIoT, da dort alle bestehenden Topics mit den jeweiligen Policys eingetragen sind. Die Suche und Filterung besteht bereits und ist dadurch eine Implementierungsaufgabe und steht weniger in der Konzeptionellen Frage.

Durch die komplexere Struktur einer Policy wurde als Anforderung an die Policy Lesbarkeit und Leichtgewichtigkeit gestellt. Die Leichtgewichtigkeit bezieht sich zusätzlich auf die Anforderung der TDLIoT. Hier gibt es Beschränkungen der Umsetzung, da die Komplexität an dem Aufbau der Policy hängt. Deshalb bezieht sich diese Anforderung auf die allgemeine Definition der Policy. Dafür wurde die Kategorien entworfen und jeder Policy Type muss einem dieser Kategorien untergeordnet werden. Das erlaubt eine einfache Übersicht. Zum anderen besteht jeder Policy Type aus einer beliebigen Anzahl von Schlüssel-Wert-Paaren, was für ein einfaches Verständnis der einzelnen Policy Types führt.

5 Protoypische Implementierung

Die Umsetzung der Konzeption wird mithilfe eines Prototypen überprüft. Im folgenden Kapitel wird die Implementierung anhand verschiedener Szenarien erläutert. Jede neue Funktionalität und Komponente die durch die Erweiterung hinzugefügt oder angepasst werden muss wird innerhalb der Umgebung der TDLIoT dargestellt. Zu Beginn wird ein Überblick über die erweiterte Architektur der TDLIoT gegeben. Danach folgen die einzelnen Szenarien. Das erste Szenario ist die Erstellung eines Policy Types im Policy Repository durch einen Policy Provider. Dieser Policy Type wird vom Policy Expert überprüft und für die Verwendung der Policy Collection freigegeben. Das Bereitstellen eines neuen Policy Types kann danach bei der Erstellung einer neuen Topic Beschreibung integriert werden. Nachdem ein neues Topic in der TDLIoT vorhanden ist wird die Verifizierung der Policy Types angestoßen. Zum Schluss wird die erweiterte Such- und Filterfunktionalität, die zur Findung von Topics bestimmter Policy Types und anderen Attributen angepasst wurde, vorgestellt. Durch diese Szenarien wird der volle Funktionsumfang der erweiterten TDLIoT erläutert.

5.1 Architekturüberblick

Für die Umsetzung des Konzepts aus Kapitel 4 müssen der TDLIoT neue Komponenten hinzugefügt und die Bisherigen angepasst werden. Um den Überblick für diese Integrierung zu schaffen wird in diesem Abschnitt eine erweiterte Architektur der TDLIoT dargestellt. Abbildung 5.1 zeigt wie die verschiedenen Komponenten in Verbindung stehen. In der ursprünglichen TDLIoT-Architektur existieren bereits die Komponenten Topic Catalog, Server und Website. Der Topic Catalog enthält alle durch die TDLIoT erstellten Topics und ist durch eine *mongoDB* Datenbank realisiert. Die Server-Komponente stellt eine REST API zur Erstellung, Änderung und Filterung von Topics bereit. Die Webseite ist ein alternativer Weg, um Topics in die Datenbank zu speichern und dem Nutzer eine einfachere Bedienung zu ermöglichen. Die REST API des Servers kann vom Topic Provider und Topic Consumer unabhängig von der Webseite genutzt werden. Durch die Erweiterung um Policys sind die Komponenten Policy Collection und Policy Repository hinzugekommen. Die Policy Collection ist in die bisherige Webseite integriert und ist eine Sammlung von Policy Types für die Erstellung einer Topic Beschreibung und Übersicht für Topic Provider sowie Consumer. Das Policy Repository ist eine Dateistruktur, bereitsgestellt auf *GitHub*[1] und

[1] https://github.com/lehmansn/TDLPolicy

lässt eine Versionierung der bisherigen Policy Types zu. Zusätzlich kann die Abgrenzung von neuen Policy Types und verifizierten Policy Types durch verschiedene Zweige der Versionierung umgesetzt werden.

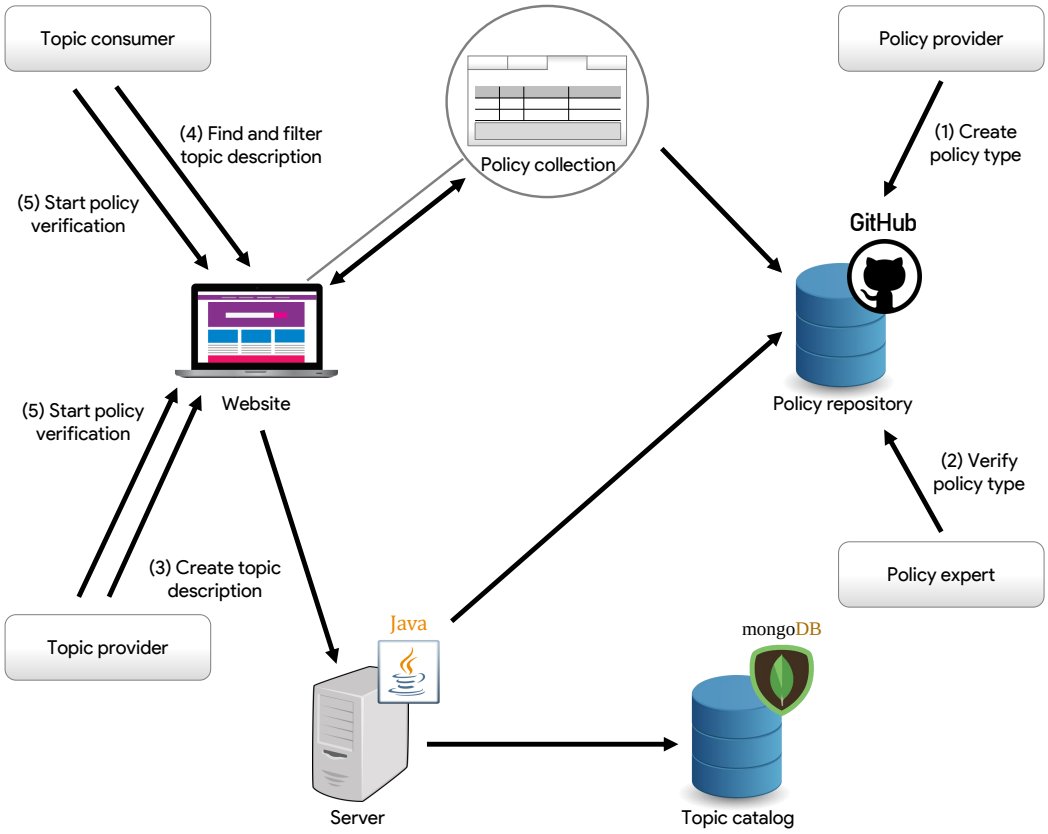

Abbildung 5.1: Architektur der TDLIoT und neue Szenarien durch die Policy Erweiterung

Zusätzlich zeigt Abbildung 5.1 welche Rollen mit den jeweiligen Komponenten interagieren und die fünf Szenarien die durch die Erweiterung um Policys angepasst wurden. Ein Policy Provider hat die Zugriffsberechtigung auf das GitHub-Projekt, um neue Policy Types zu erstellen. Der Policy Expert hat die Möglichkeit neue Policy Types zu überprüfen und als verifiziert zu markieren, damit die Policy Collection neue Policy Types in ihre Sammlung aufnehmen kann. Die farbigen Pfeile stellen Ablaufszenarien dar, die die Implementierungen und zusätzlichen Funktionalitäten erklären. Szenario (1) ist das Hinzufügen eines Policy Types zum Policy Repository durch einen Policy Provider. Dieser neue Policy Type wird danach in Szenario (2) auf Qualität und Duplikate, durch den Policy Expert, überprüft. Das umfangreichste Szenario (3) ist die Erstellung einer Topic Beschreibung mit Einbeziehung von Policys. Die Daten werden dann im JSON Format an den Server geschickt und auf Richtigkeit überprüft. Diese Überprüfung bezieht sich auf das JSON Schema und die Validierung der Policy Types, ob alle Attribute vorhanden sind und diese mit den richtigen Datentypen beschrieben wurden. Danach wird die gesamte Topic Beschreibung in die

mongoDB eingetragen. Danach folgt das Szenario (4), in diesem erfolgt das Anstoßen einer Verifizierung, die die Policy eines Topics übeprüft, um herauszufinden ob die Angaben des Topic Providers eingehalten werden. Um als Topic Consumer das passende Topic bezüglich Policys zu finden wurde die Such- und Filterfunktionalität angepasst, dies ist das letzte Szenario (5).

5.2 Erstellung und Überprüfung von Policy Types

Ein Policy Provider hat über ein GitHub Projekt[2], das Policy Repository, die Möglichkeit neue Policy Types für seinen Anwendungsfall zu definieren. Ein Repository ist ein Dateiverzeichnis welches Versionierung zulässt. Zusätzlich ist es möglich Zweige der Datenstruktur zu erstellen. Das bedeutet, dass in einem Zweig Dateien existieren können, die im Anderen nicht vorhanden sind. Dennoch kann durch einfache Operationen ein Abgleich der verschiedenen Zweige erfolgen. Das erlaubt es die Rolle des Policy Expert sinnvoll umzusetzen, da es einen Zweig für Policy Provider gibt, die eigene Policy Types hinzufügen können und einen anderen Zweig für Policy Experts, die den Zweig der Policy Provider einsehen, die darin befindlichen Policy Types überprüfen und nach erfolgreichem Abschließen dieser Verifizierung den Policy Type ihrem Zweig hinzufügen können. Abbildung 5.2 zeigt den relevanten Ausschnitt der Architektur für die Erstellung und Überprüfung eines Policy Types. Beide Szenarien spielen sich nur im Bereich des Policy Repositorys ab und sind soweit unabhängig voneinander.

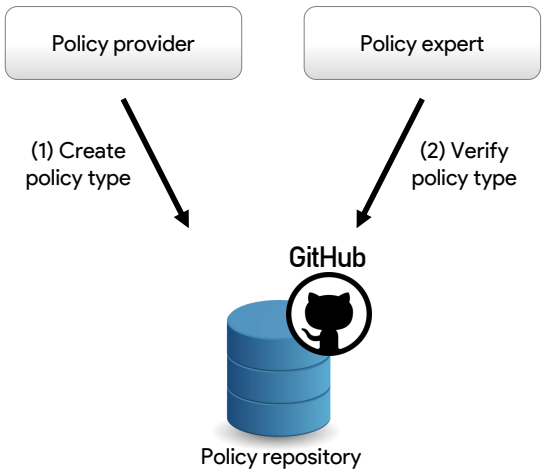

Abbildung 5.2: Szenarien der Erstellung und Verifizierung, durch den Policy Expert, eines Policy Types

[2] https://github.com/lehmansn/TDLPolicy

Für die Erstellung legt der Policy Provider eine Datei im JSON Format an und benennt diese nach dem neuen Policy Type. Der Name der Datei muss einmalig sein und es darf keine andere Datei mit dem gleichen Namen im Repository vorhanden sein. Danach befüllt er diese Datei mit allen Angaben die die Policy Collection zum Anzeigen eines Policy Types benötigt. Um zwischen der Beschreibung eines Policy Types im Repository und der Beschreibung eines Policy Types in der TDLIoT unterscheiden zu können, wird der Policy Type im Policy Repository als Metamodell benannt, weil es die Beschreibung eines Policy Types der TDLIoT beinhaltet. Der Aufbau eines Policy Types besteht aus folgenden Attributen:

policy_type: Dieses Attribut enthält, wie der Name der JSON Datei, die Bezeichnung des Policy Types.

description: Die Beschreibung ist eine kurze Erläuterung für welchen Anwendungsfall dieser Policy Type zu verwenden ist, als Unterstützung zum Verständnis des Policy Types für Topic Provider und Consumer.

policy_category: Die Kategorie ist der Indikator für die Eingliederung der Policy in der TDLIoT. Dafür gibt es zwei Möglichkeiten, entweder ist der Policy Type auf das Topic bezogen oder auf die Nachricht die vom Topic publiziert wird. Der Policy Provider muss sich im Klaren sein, für welchen Zweck der neue Policy Type dienen soll und muss dementsprechend „topic“ oder „message“ eintragen.

input: Jedes Modell kann beliebig viele Attribute besitzen, deshalb werden in diesem Attribut alle Attribute des Modells, bis auf das Attribut „policy_type“, als Liste eingetragen. Diese Liste enthält die Metamodelle der Attribute des Modells. Das Metamodell eines Attributs besteht aus mindestens drei Angaben. Zum einen aus dem „value“, dieser enthält die Bezeichnung des Attributs Zum anderen der „datatype“, welcher anzeigt welchen Datentyp dieses Attribut enthalten kann und zum Schluss aus der „description“, was die Beschreibung des Attributs für Topic Provider und Consumer ist. Im Falle, dass der Datentyp des Attributs nur spezielle Begriffe erlaubt, muss ein weiteres Attribut dem Metamodell hinzugefügt werden. Dieses Attribut heißt „enum“ und beschreibt eine Liste aller Begriffe die für das Attribut des Modells zugelassen sind.

example: Im Policy Type ist, zum Verständnis der Attribute des Modells, ein exemplarisches Policy Type Modell enthalten.

Listing 5.1 zeigt die Beschreibung des Policy Types Intervall im Policy Repository. Es besteht aus den gerade aufgezählten Attributen. In diesem Fall zeigt das letzte Attribut des Metamodells, dass es möglich ist ein Intervall von einer Sekunde zu definieren. Dafür wurde dem Policy Type einen Namen gegeben, „fast interval“, der dem Provider die Möglichkeit gibt ein weiteres Topic anzubieten, das beispielsweise „long interval“ heißt und die selben Daten im langsameren Intervall liefert. So kann eine Abgrenzung zwischen zwei Topics geschaffen werden. Der Wert und die Einheit beziehen sich auf das Intervall und definieren in diesem Beispiel eine Sekunde. Das Metamodell der Attribute zeigt, welche Intervalle

mit diesem Policy Type umsetzbar sind. Da die Einheit von Millisekunden bis hin zu Tagen sein kann und eine einzelne Ganzzahl die Länge bestimmt kann ein Intervall zwischen einer Millisekunde und mehreren Tagen definiert werden.

Listing 5.1 Policy Type Intervall

```
{
  "policy_type": "interval",
  "description": "Defines the time between two published values",
  "policy_category": "message",
  "input": [ {
      "value": "name",
      "datatype": "string",
      "description": "Unique name of this policy"
    }, {
      "value": "value",
      "datatype": "int",
      "description": "Value as integer"
    }, {
      "value": "unit",
      "datatype": "enum",
      "description": "Time unit of the value",
      "enum": [
        "ms",
        "sec",
        "min",
        "hour",
        "day"
      ]
    }
  ],
  "example": {
    "name": "fast interval",
    "value": "1",
    "unit": "sec"
  }
}
```

Für die Überprüfung eines Policy Types im Policy Repository ist der Policy Expert zuständig. Er hat die Berechtigung einen Policy Type vom ursprünglichen Zweig, in dem alle Policy Provider neue Policy Types hinzufügen, dem Zweig der verifizierten Policy Types zu portieren. Dafür müssen zwei Aspekte überprüft werden, zum einen, ob ein ähnlicher Policy Type mit anderen Namen bereits existiert. Wenn dies der Fall ist muss der Policy Expert entscheiden, ob die beiden Anwendungsfälle zu unterschiedlich sind, um sie in einen Policy Type zusammenzuführen oder ob durch Erweiterungen des bisherigen Policy Types eine Zusammenführung die beste Lösung bringt. Zum anderen, ob die Attribute für den Anwendungsfall sinnvoll gewählt wurden. Dabei wird überprüft ob unnötige Attribute

vorhanden sind oder die Beschreibungen zu ungenau oder kompliziert definiert sind. Wird ein bereits verifizierter Policy Type durch eine Policy Provider geändert muss der Policy Expert die komplette Verifizierung erneut durchführen.

5.3 TDLIoT Erstellung

Der Ablauf der Erstellung einer TDLIoT ist für den Topic Provider vom Grundprinzip gleich geblieben. Bisher kann ein Topic Provider mit den Daten seines Topics eine TDLIoT auf der Webseite erstellen. Beim Speichern wird die TDLIoT im JSON Format an den Server geleitet und dieser integriert die Daten in den Topic Catalog. Danach kann die Webseite die neu erstellte TDLIoT anzeigen und dadurch von einem Topic Consumer gefunden werden.

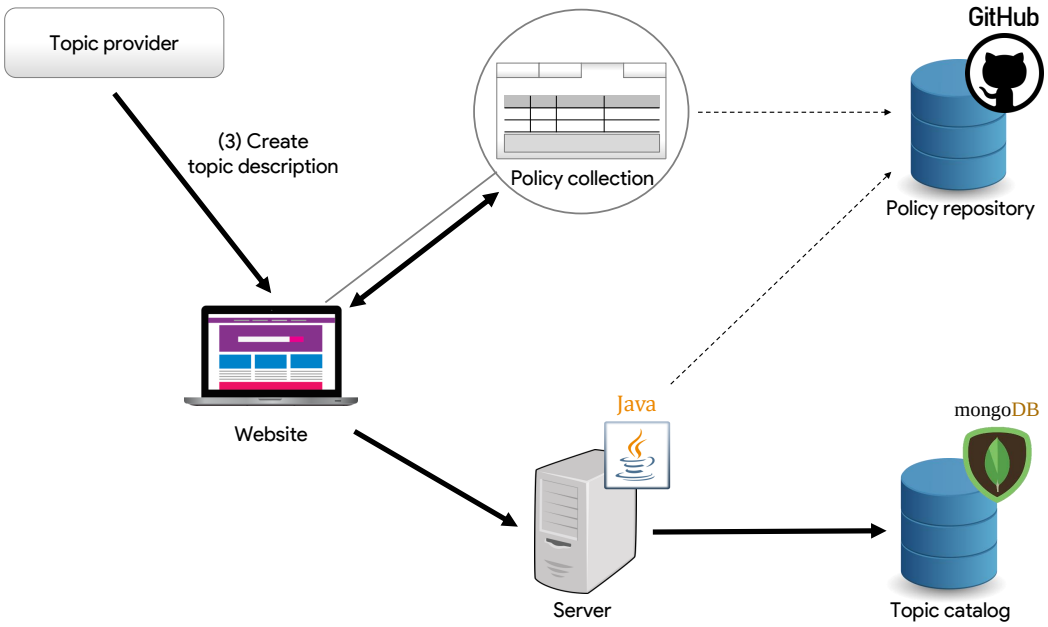

Abbildung 5.3: Szenario der Erstellung einer TDLIoT mit der Erweiterung von Policy

Abbildung 5.3 stellt den erweiterten Ablauf der Erstellung einer TDLIoT dar. Die gestrichelten Pfeile stellen den Informationsaustausch zum Policy Repository dar, welches unabhängig vom Topic Provider abläuft. Der eigentliche Ablauf einer Erstellung wird mit den dickeren grünen Pfeile dargestellt. Der Topic Provider kann wie bisher auf der Webseite die Daten seines Topics für die TDLIoT eintragen. Durch die Erweiterung von Policys kann er mithilfe der Policy Collection, die ihre verifizierten Policy Types aus dem Policy Repository lädt, passende Policy Types für sein Topic suchen und seiner TDLIoT hinzufügen. Die gesamte Beschreibung des Topics wird beim Speichern im JSON Format an den Server gesendet. Dieser speichert nicht mehr direkt die TDLIoT in der mongoDB,

sondern führt einige Überprüfungen durch. Zum einen wird das JSON Schema der TDLIoT auf Richtigkeit und Vollständigkeit validiert. Zum anderen werden die Policy Types mithilfe der Beschreibung im Policy Repository überprüft, damit alle Attribute eingetragen und dessen Datentypen richtig eingehalten wurden. Danach wird die TDLIoT in die Datenbank gespeichert.

5.3.1 Eintragung der Daten eines Topics

Zu Beginn muss der Topic Provider die Eckdaten seines Topics in der Webseite eintragen. Bisher waren das die Attribute der TDLIoT ohne Policys (z.B. *owner*, *path*, *protocol*) [FHB+18], die Beschreibung, wie auf das Topic zugegriffen werden kann, was für Daten bereitgestellt werden und wem diese Daten gehören, also funktionale Anforderungen. Mit der Erweiterung um Policys hat der Policy Provider die Möglichkeit die Sammlung der verifizierten Policy Types in der Policy Collection einzusehen und den, für seinen Anwendungsfall passenden, Policy Type seiner Topic Beschreibung hinzuzufügen. Abbildung 5.4 zeigt die prototypische Implementierung der Policy Collection, welche in die Webseite integriert ist, damit bei der Erstellung der TDLIoT die Übersicht vorhanden bleibt. Sie besteht aus verschiedenen Seiten die über die Reiter am oberen Rand gewechselt werden können. Jede Seite enthält eine detaillierte Beschreibung eines Policy Types. Die Informationen der Beschreibung bezieht die Policy Collection aus den verifizierten Policy Types des Policy Repositorys. In der letzten Spalte der Tabelle kann der Topic Provider die Daten seines Topics eintragen. In Abbildung 5.4 ist der Policy Type Intervall zu sehen wobei ein Intervall von einer Stunde definiert wurde. Beim Drücken des Buttons „Add new Policy Type“ werden die eingetragen Daten übernommen und in das Feld „Topic Policy Types“, welches sich über der Policy Collection befindet, eingefügt. Der Policy Type der Kosten ist bereits vorhanden und ist dementsprechend ein Teil der aktuell erstellten TDLIoT. Sie besagt, dass das Anmelden an das Topic, für den Topic Consumer, pro Monat 0,99€ kostet. Jeder Policy Type kann nur einmal dem Topic angefügt werden. Um Variationen unterschiedlicher Policy Types für ein Topic zu definieren müssen verschiedene Beschreibungen eines Topics erstellt werden. Durch das Speichern der gesamten Beschreibung werden die Daten im JSON Format über die REST Schnittstelle an den Server übermittelt.

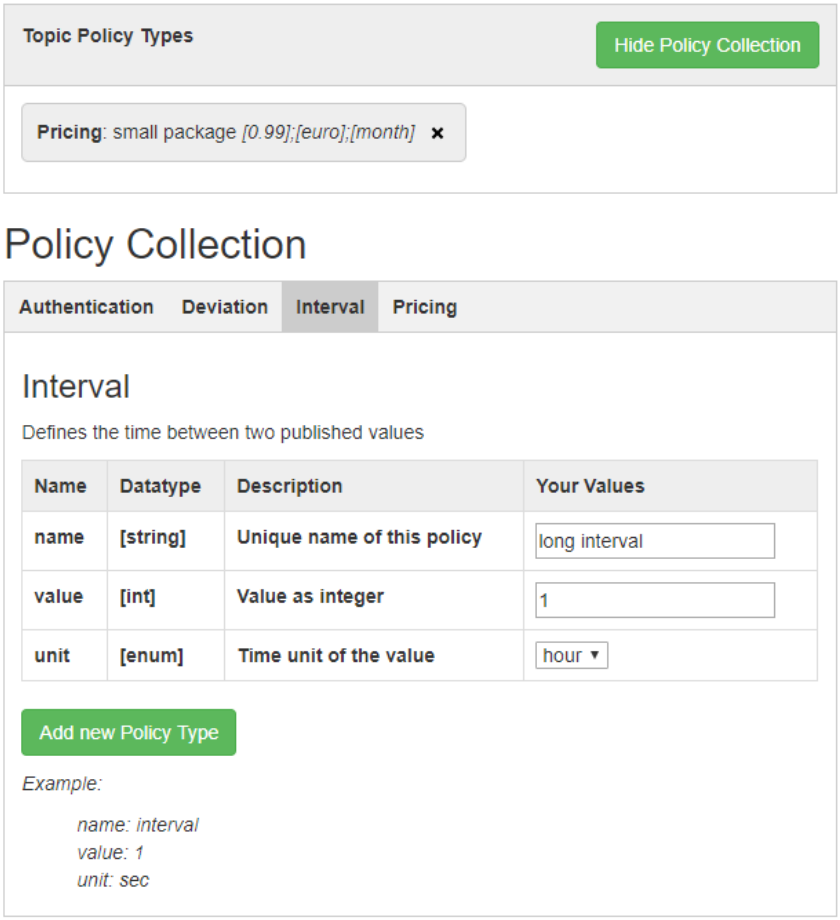

Abbildung 5.4: Umsetzung der Policy Collection im Prototypen mit Beispiel eines Policy Types

5.3.2 Validierung und Speicherung durch den Server

Die REST Schnittstelle des Servers bietet einem Topic Provider an, ohne die Nutzung der Webseite ein Topic zu erstellen. Das ermöglicht die Integrierung der Erstellung und Änderung der TDLIoT in andere Systeme. Die Übersicht der Policy Collection ist von dieser Funktionalität ausgeschlossen und kann nur über die Webseite betrachtet werden. Wird eine neue TDLIoT an den Server übermittelt starten zwei verschiedene Validierungen. Die erste Validierung betrifft jede TDLIoT-Beschreibung und validiert gegen das JSON Schema. Dieses Schema gibt die Struktur vor und zeigt welche Attribute der TDLIoT im JSON Format benötigt werden. Daraus folgt, dass nicht optionale Attribute vorhanden sein müssen und jeder Policy Type mindestens die Attribute *policy_type* und *name* enthalten muss. Es ist eine reine syntaktische Überprüfung, die Validierung der Semantik ist darin nicht enthalten. Die zweite Validierung wird nur bei einer TDLIoT durchgeführt, die Policy Types enthält. Sie ist wie die Validierung des JSON Schemas eine syntaktische Überprüfung im Bezug auf die Definition des Policy Types im Policy Repository. Es wird uberprüft ob

alle Attribute vollständig und mit den richtigen Datentypen beschrieben wurden. Bei einem fehlerhaften Abschließen einer der beiden Validierungen wird dem Topic Provider eine Fehlermeldung übermittelt, die ihm zeigt, an welcher Stelle der Fehler aufgetreten ist. Bei erfolgreichem Abschließen kann die TDLIoT in den Policy Catalog gespeichert werden. Es ist möglich eine vorhandene TDLIoT anzupassen. Dies kann beispielsweise der Pfad des Topics, Änderung des Protokolls zur Übermittlung oder eine Anpassung eines Policy Types sein. In diesem Fall ist der Ablauf zur Speicherung dieser Änderung im Policy Catalog der Gleiche wie bei der Neuerstellung beschrieben.

5.4 Policy Type Verifizierung

Ein Topic kann über viele Jahre im Einsatz sein. Über einen längeren Zeitraum ändern sich oftmals die Umgebungsverhältnisse. Das kann beispielsweise der Sensor sein, welcher nach einiger Zeit kaputt geht und gewechselt werden muss oder das Hinzufügen einer Authentifizierung, um den Zugriff auf das Topic einzuschränken. In jedem Fall muss der Topic Consumer herausfinden können ob die Beschreibung eines Topics in der TDLIoT der Wahrheit entspricht. Für diese Überprüfung wurde eine neue Funktionalität der TDLIoT hinzugefügt. Sie zeigt ob ein Topic mit dem gewählten Pfad erreichbar ist und welcher Policy Type eines Topics die Definition einhält. Zudem kann eine neue Verifizierung angestoßen werden, um aktuelle Informationen über den Zustand des Topics zu erhalten.

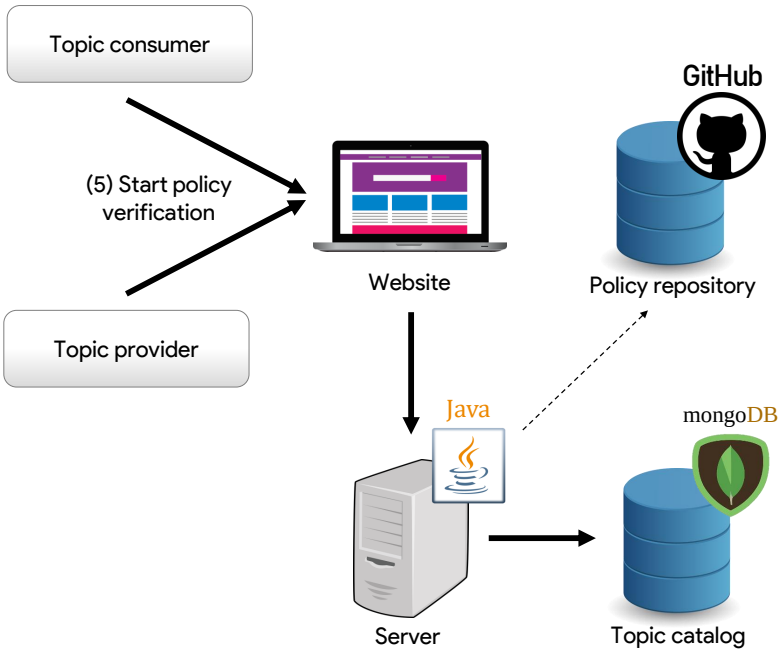

Abbildung 5.5: Szenario der Verifizierung der Policy eines Topics

Das Szenario einer Verifikation im System der TDLIoT ist in Abbildung 5.5 dargestellt. In der Regel wird ein Topic Consumer eine Verifikation starten, um zu prüfen ob die Policy Types eines Topics der Wahrheit entsprechen. Das Starten der Verifizierung kann von der Webseite oder direkt durch Ansprechen der REST Schnittstelle des Servers erfolgen. Auf dem Server liegen die Implementierung der Verifizierung einzelner Policy Types. Eine Überprüfung wird immer für ein einzelnes Topic angestoßen und dadurch alle Policy Types dieses Topics verifiziert. Ist keine Implementierung der Verifikation eines Policy Types auf dem Server vorhanden, wird der Zustand *unbekannt* als Resultat zurückgegeben. Kann eine Überprüfung durchgeführt werden, ist die Dauer der Verifikation abhängig vom Policy Type. Bei der Prüfung der Authentifizierung ist dies beispielsweise sehr kurz, da versucht wird ohne Zugriffsberechtigung bei dem Topic anzumelden. Ist dies möglich ist die Verifikation fehlgeschlagen. Dagegen ist bei der Prüfung des Intervalls mehr Zeit notwendig, da dies vom Intervall des Topics abhängt. Es wird die Zeit zwischen mehreren publizierten Werten gemessen wird und überprüft ob jeder Intervall der Definition aus der Policy Beschreibung entspricht. Nach der Verifikation speichert der Server das Ergebnis in der Datenbank und wird im Datenmodell des überprüften Topics integriert, so sind die Informationen des gesamten Topic an einem Ort gespeichert und zugänglich.

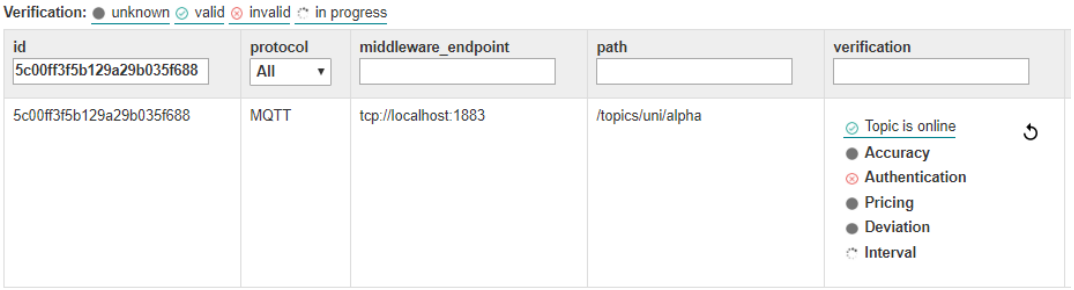

Abbildung 5.6: Die vier möglichen Zustände der Verifikation eines Policy Types und ein Topic mit laufender Verifizierung.

Abbildung 5.6 stellt die Visualisierung der Policy Type Verifizierung im Prototyp dar. Über der Tabelle ist eine Legende aller möglichen Zustände einer Verifikation abgebildet. Diese erklärt dem Nutzer die Bedeutung der Symbol in der darunterliegenden Tabelle. Das Topic zeigt in der Spalte *verification* unterschiedliche Verifizierungen mit ihren Zuständen. Der oberste Eintrag in dieser Spalte ist immer die Information, ob das Topic erreichbar ist. Dafür sind nur zwei Zustände möglich, *valid* wenn das Topic einen korrekten Pfad besitzt und auf dem System des Topics das Protokoll unterstützt wird oder *invalid* wenn diese Anforderungen nicht eingehalten werden können. Alle anderen Einträge beziehen sich auf die Policy Types des Topics. In diesem Beispiel sind die Policy Types Genauigkeit, Kosten und Abweichung mit dem Zustand *unknown* versehen. Das bedeutet sie sind nicht verifizierbar und es kann keine Aussage über die Richtigkeit der Angaben gemacht werden. Der Policy Type des Intervalls wird gerade überprüft und resultiert in einem erfolgreichen oder fehlgeschlagenen Ergebnis. Die Verifikation der Authentifizierung ist abgeschlossen und fehlgeschlagen, das bedeutet, dass der Zugriff auf das Topic ohne Authentifizierung möglich war und der Definition des Policy Types widerspricht.

Listing 5.2 Beispiel für das Datenmodell einer Verifikation

```
"verification": {
  "Accuracy": {
    "current": "unknown"
  },
  "Authentication": {
    "current": "invalid"
    "last": "valid",
    "timestamp": "30-11-2018_12:04:34"
  },
  "Pricing": {
    "current": "unknown"
    "last": "unknown",
    "timestamp": "30-11-2018_12:04:34"
  },
  "Deviation": {
    "current": "unknown"
  },
  "Interval": {
    "current": "in progress"
    "last": "valid",
    "timestamp": "30-11-2018_12:04:34"
  }
}
```

In der Datenbank werden die Zustände der Policy Types im JSON Format wie in Listing 5.2 abgespeichert. Jede Verifizierung eines Policy Types hat einen aktuellen Zustand. Falls es bereits eine Überprüfung gab enthält sie zusätzlich den letzten Zustand mit dem dazugehörigen Zeitstempel des Startzeitpunktes der Überprüfung.

5.5 Erweiterte Topic Suchfunktion

Die bisherige Such- und Filterfunktionalität war eine reine Schlüssel-Wert-Paar überprüfung. Da die TDLIoT im JSON Format gespeichert wird, werden die, von der Hierarchie, obersten Attribute auf Gleichheit bezüglich des Suchbegriffs überprüft. Die Such- und Filterfunktion wurde soweit erweitert, dass sie durch die Verschachtelung der Attribute iteriert und nach einer Übereinstimmungen des Namens eines Attributes den Wert mit dem Suchbegriff vergleicht. Zusätzlich soll es möglich sein Kombinationen aus verschiedenen Vergleichsoperatoren zu suchen und filtern.

Der Ablauf dieses Szenarios ist in Abbildung 5.7 dargestellt. Dieser ist im Vergleich zum Ablauf vor der Erweiterung durch Policy gleich geblieben. Durch die Suchfunktionen der Webseite oder der REST Schnittstelle können Topic Provider und Consumer nach Topics in der TDLIoT suchen. Dieser leitet die Suche an den Policy Catalog weiter und gibt die Suchergebnisse zurück. Die erweiterte Funktionalität läuft im Server ab, denn der Server

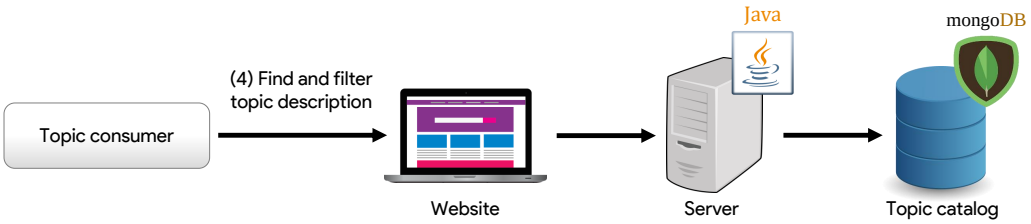

Abbildung 5.7: Szenario der Such- und Filterfunktion eines Topics

baut aus der Anfrage des Nutzers eine komplexe Anfrage, die es erlaubt in den tieferen Ebenen der Datenstruktur zu suchen. Dadurch werden über das Netzwerk nur die relevanten Ergebnisse übermittelt. Wenn der Server die Filterung selbst durchführen würde, könnte es bei großen Datenmengen und vielen Anfragen für die Schnittstelle, von Server zur Datenbank, zur Überlastung kommen.

Listing 5.3 Beispiel einer Suchanfrage

```
{
    "protocol": {"$eq": "MQTT"},
    "cost": {
        "$gt": 0.5,
        "$lt": 5
    },
    "deviation": {"$lt": 1}
}
```

Eine beispielhafte Suchanfrage eines Topic Providers oder Consumers ist in Listing 5.3 abgebildet. In diesem Format können beliebig viele Suchkriterien hinzugefügt werden. Die erweiterte Suchfunktion stellt vier verschiedene Vergleichsoperatoren bereit, diese sind durch mongoDB [Mon18] spezifiziert. Es sind die Operatoren größer „$gt“, kleiner „$lt“, gleich „$eq“ und ungleich „$ne“. Durch die Kombination der Operatoren aus größer und kleiner kann zudem ein Suchbereich definiert werden. Der Server verwendet bei einer Anfrage, wie in Listing 5.3, jedes Attribut als Suchkriterium und filtert, in den einzelnen Ebenen des JSON Objekts, nach dem passenden Schlüssel zu einem Attribut. Im Beispiel des Suchkriteriums *cost* wird in der gesamten TDLIoT nach einem Attribut mit Namen *cost* gesucht. Dabei können mehrere Policy Types dieses Attribut haben oder auch bei weiteren Erweiterungen der TDLIoT kann dieses Attribut in unterschiedlichem Kontext aufgetreten. In jedem Fall zählt die Übereinstimmung des Schlüssels eines Attributs als erfolgreiche Suche. Danach wird der Wert der sich darin befindet mit den Operatoren, die in der Suchanfrage stehen abgeglichen. In diesem Beispiel wird der Wert auf größer 0,5 und kleiner fünf überprüft. Ist dies für eine TDLIoT zutreffend, wird sie an den Server zurückgegeben. Sind mehrere Suchkriterien in einer Anfrage vorhanden, wie in Listing 5.3, müssen für die gesuchten Topics alle Kriterien gelten. Das ermöglicht eine exakte Filterung und Abgrenzung zwischen den Topic Beschreibungen in der Datenbank.

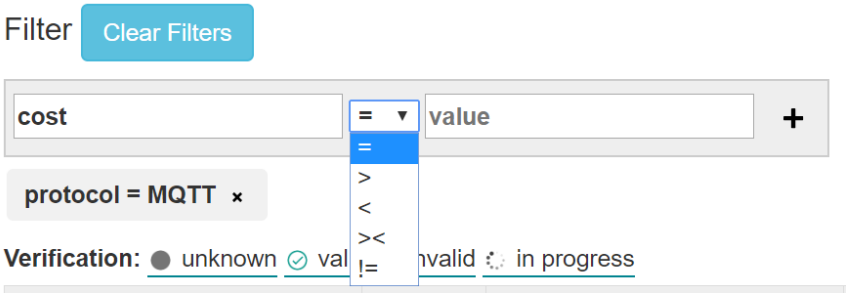

Abbildung 5.8: Such- und Filterfunktion auf der Webseite des Prototyps

Die Webseite bietet dem Benutzer diese Such- und Filterfunktionalität auf der Benutzeroberfläche. Dadurch muss er das Format einer Suchanfrage, um eine valide Anfrage mit unterschiedlichen Suchkriterien zu stellen, nicht kennen. Abbildung 5.8 zeigt den Aufbau der Filterfunktion auf der Webseite. Im ersten Feld wird der Schlüssel des gesuchten Attributs eingetragen. In der Mitte ist die Auswahl des Suchoperators und am Ende das Feld, für den Wert des gesuchten Attributs. Bei einer Bereichsanfrage entsteht für den zweiten Wert ein weiteres Feld um den Bereich nach unten und oben einzugrenzen. Alle hinzugefügten Suchkriterien werden darunter abgebildet, damit der Benutzer sieht, nach welchen Kriterien gefiltert wird.

6 Zusammenfassung und Ausblick

In diesem Kapitel wird die Masterarbeit zusammengefasst und es werden zukünftige Arbeiten beschrieben. Dies umfasst die Konzeption und die Implementierung der Erweiterung der TDLIoT durch Policys. Zum Schluss wird eine Diskussion über die entstandene Fragestellung und Anmerkungen der bisherigen Umsetzung geschildert. Dazu gehört ein Ausblick auf zukünftige Aufgaben für die erweiterte TDLIoT. Das bedeutet, dass Lösungsansätze für Diskussionspunkte vorgestellt und Aufgaben, welche potentiell nach dieser Arbeit erfolgen können aufgezeigt werden.

6.1 Zusammenfassung

Das Ziel dieser Arbeit war die Erweiterung der TDLIoT durch Policys. Das Ergebnis wurde als Prototyp implementiert. Als erstes wurde das Konzept der Policy für die TDLIoT entwickelt. Zum einen besteht dies daraus, dass eine Policy aus unterschiedlichen Policy Types besteht, welche nichtfunktionale Anforderungen, SLA oder QoS abbilden. Dazu gehört der strukturelle Aufbau eines Policy Types und welchen Anforderungen diese unterliegen. Zum anderen gehört dazu, die Integration der Policy in das TDLIoT Gesamtmodell. Durch diese Anpassungen mussten die Komponenten und Rollen der TDLIoT erweitert werden. Bisher bestand die TDLIoT aus Topic Provider, Topic Consumer und Topic Catalog. Als neue Rollen wurden Policy Provider, welcher neue Policy Types erstellen kann, und Policy Expert, welcher überprüft die Policy Types des Policy Providers auf ihre Qualität der Aussagekraft und Überschneidungen mit anderen Policy Types, hinzugefügt. Die Rolle des Policy Expert kann momentan nur von einem Menschen übernommen werden und dieser reguliert als einzige Instanz die Qualität der Policy Types. Zusätzlich wurde die Komponente Policy Repository hinzugefügt. Sie enthält alle Policy Types der Policy Provider und überprüften Policy Types der Policy Experts. Als letzte Komponente wurde die Policy Collection entworfen, die als Schnittstelle für den Topic Provider zum Policy Repository fungiert. Die Policy Collection enthält alle vom Policy Expert überprüften Policy Types und stellt die Informationen eines Policy Types detailliert in einer Übersicht dar, damit der Topic Provider die passenden Policy Types für sein Topic finden und der Beschreibung des Topics hinzufügen kann. Der letzte wichtige Baustein der Erweiterung ist die Verifizierung der Policy Types, die in einer erstellten Topic Beschreibung in der TDLIoT vorhanden ist. Damit kann der Topic Consumer überprüfen ob die Angaben des Topic Providers eingehalten werden. Nach der konzeptionellen Phase wurde dem bisherigen Prototypen die neuen Modelle und Funktionalitäten hinzugefügt.

Die Policy Collection ist ein Teil der Webseite der TDLIoT geworden, um die Übersicht der Policy Types während der Erstellung einer Topic Beschreibung einsehen zu können. Gleichzeitig kann durch die Policy Collection eine Instanz eines Policy Types definiert und der Topic Beschreibung hinzugefügt werden. Die Verifikation wurde im Server realisiert und kann, durch den Zugriff auf die Definitionen der Policy Types im Policy Repository, nach der Erstellung einer Topic Beschreibung gestartet werden. Während der Erstellung wird die Beschreibung, bevor sie im Topic Catalog gespeichert wird, syntaktisch validiert. Wenn ein Topic Consumer nach Topics in der TDLIoT sucht, bringt die Erweiterung durch Policys neue Filtermöglichkeiten. Diese Such- und Filterfunktionalität wurde in der TDLIoT angepasst, um dem Benutzer die bisherige Benutzbarkeit weiterhin zu bieten, aber die Filterung in der Datenbank komplexer zu gestalten, um durch die Verschachtelung von Topic und Policy suchen zu können.

Die Vorteile dieses Aufbaus der TDLIoT sind die Generizität und Erweiterbarkeit des Policy Modells sowie die Kontrollstruktur von neuen Policy Types. Durch die Unterteilung des Modells der Policy in zwei Kategorien, ist es möglich eine Strukturiertheit zu gewähren ohne die Erweiterbarkeit einzuschränken. Policy Types werden aus verschiedenen Kontexten definiert und sind dadurch oftmals verschieden aufgebaut. Das Modell einer Policy Type bildet diese Generizität ab, indem beliebig viele Attribute hinzugefügt werden können. Die Basis der Attribute aus *policy_type* und *name* ist bei allen Policy Types gleich und erlaubt die Unterscheidung zwischen Policy Types an den beiden Attributen. Für die TDLIoT ist diese Artenvielfalt eine Vorteil zur Filterung von Topic Beschreibungen und gibt dem Topic Consumer neue Funktionalitäten, um nach seinem gewünschten Topic zu filtern. Die Kontrollstruktur des Policy Repositorys, mit dazugehöriger Policy Collection, ermöglicht eine flexible Qualitätssicherung, die dem Topic Provider hilft passende Policy Types für seine Topic Beschreibung zu finden. Durch die Möglichkeit Verifikationen zu jedem Zeitpunkt zu starten garantiert zudem die Einhaltung der Policy von einem Topic Provider und liefert dem Topic Consumer eine grundlegende Sicherheit des Wahrheitsgehaltes eines Policy Types.

6.2 Ausblick

Im Laufe der Arbeit sind verschiedene Fragestellungen und Limitationen aufgetreten. Diese werden im Folgenden dargestellt und vorhandene Lösungsansätze aufgezeigt. Bei einem steigenden Bekanntheitsgrad der TDLIoT werden mehr Policy Types erstellt und dadurch mehr Policy Experts benötigt, um diese zu überprüfen, damit die Dauer einer Überprüfung die Nutzbarkeit der TDLIoT nicht einschränkt. Hier wird eine allgemeine Definition zur Qualitätssicherung benötigt, die besagt, an welchen Standards und Richtlinien sich die Policy Experts bei der Überprüfung halten müssen. Dies kann in verschiedenen Formen (z.B. in Schrift, Video, Audio) für einen neuen Policy Expert zugänglich gemacht werde, um eine einheitliche Überprüfung unterschiedlicher Policy Experts zu gewährleisten. Eine weitere Einschränkung ist beispielsweise die Suche nach standortspezifischen Topics.

Hier muss der Benutzer wissen wie das Attribut *location value* mit dem dazugehörigen *location type* definiert wurde. Das kann eine Koordinate, Stadtname oder eine andere begriffliche Abkürzung sein. Für die Filterung gibt es keinen Zusammenhang zwischen den unterschiedlichen Positionsarten und kann deshalb nicht in Relation gebracht werden. Das gilt auch für andere Attribute der TDLIoT. Der Ansatz einer Ontologie nach Bermudez-Edo et al. [BEBT16] könnte eine Verbesserung der Benutzbarkeit für die Topic Consumer bringen. Dadurch können Begriffe als Einheit zusammengefasst und in der Filterung in Verbindung gebracht werden. Der TDLIoT könnten kleinere Erweiterungen helfen eine umfassendere Beschreibung zu sein. Zum einen eine Referenz zur Kontaktaufnahme zum Topic Provider für den Consumer, da im Bereich der Policy Types vertragliche Anforderungen (z.B. die Kosten) gestellt werden können, die außerhalb der TDLIoT abgehandelt werden müssen. Diese Information wird im aktuellen Modell der TDLIoT nicht bereitgestellt. Zum anderen ein Bewertungssystem für ein Topic im Bezug auf Zugriffszahlen und Zufriedenheit. Mit diesen Informationen kann ein Topic Consumer leichter einschätzen wie zuverlässig das Topic des Anbieters ist und inwiefern die anderen Consumer zufrieden damit sind. Das würde den Topic Provider zusätzlich einen Antrieb schaffen die Topics besser zu warten und instand zu halten. Für mehr Stabilität im Bereich von Namensräume bezüglich Attributen eines Policy types könnte das bisherige JSON Schema durch JavaScript Object Notation for Linked Data (JSON-LD) erweitert werden. Damit sind *Namespaces* wie in XML umsetzbar und verhindern Konflikte bezüglich der Namen der Attribute. Das gesamte System ist als Prototyp implementiert. Durch die Erweiterung der Policys kann die Implementierung als cloud-basiertes Produkt gestartet werden.

Literaturverzeichnis

[AIM10] L. Atzori, A. Iera, G. Morabito. „The internet of things: A survey“. In: *Computer networks* 54.15 (2010), S. 2787–2805 (zitiert auf S. 19).

[BBKM15] E. Balandina, S. Balandin, Y. Koucheryavy, D. Mouromtsev. „IoT use cases in healthcare and tourism“. In: *Business Informatics (CBI), 2015 IEEE 17th Conference on*. Bd. 2. IEEE. 2015, S. 37–44 (zitiert auf S. 19).

[BEBT16] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, K. Taylor. „IoT-Lite: a lightweight semantic model for the Internet of Things“. In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. IEEE. 2016, S. 90–97 (zitiert auf S. 65).

[BS11] D. Bandyopadhyay, J. Sen. „Internet of things: Applications and challenges in technology and standardization“. In: *Wireless Personal Communications* 58.1 (2011), S. 49–69 (zitiert auf S. 15).

[CCV12] M. Collina, G. E. Corazza, A. Vanelli-Coralli. „Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST“. In: *Personal indoor and mobile radio communications (pimrc), 2012 ieee 23rd international symposium on*. IEEE. 2012, S. 36–41 (zitiert auf S. 15, 20).

[Che76] P. P.-S. Chen. „The entity-relationship model—toward a unified view of data“. In: *ACM Transactions on Database Systems (TODS)* 1.1 (1976), S. 9–36 (zitiert auf S. 22, 35).

[CNYM12] L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos. *Non-functional requirements in software engineering*. Bd. 5. Springer Science & Business Media, 2012 (zitiert auf S. 16).

[CSZS07] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc. „Automated classification of non-functional requirements“. In: *Requirements Engineering* 12.2 (2007), S. 103–120 (zitiert auf S. 31).

[FBFL14] C. Fehling, J. Barzen, M. Falkenthal, F. Leymann. „PatternPedia-collaborative pattern identification and authoring“. In: *Proceedings of PURPLSOC (Pursuit of Pattern Languages for Societal Change). The Workshop*. 2014, S. 252–284 (zitiert auf S. 40).

[FHB+18] A. C. Franco da Silva, P. Hirmer, U. Breitenbücher, O. Kopp, B. Mitschang. „TDLIoT: A Topic Description Language for the Internet of Things“. In: *International Conference on Web Engineering*. Springer. 2018, S. 333–348 (zitiert auf S. 15, 21, 24, 25, 29, 38, 44, 55).

[Gartner17] R. van der Meulen. *Gartner Says 8.4 Billion Connected "Things"Will Be in Use in 2017, Up 31 Percent From 2016*. 2017. URL: http://www.gartner.com/newsroom/id/3598917 (zitiert auf S. 15).

[GBMP13] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami. „Internet of Things (IoT): A vision, architectural elements, and future directions“. In: *Future generation computer systems* 29.7 (2013), S. 1645–1660 (zitiert auf S. 19).

[Gli07] M. Glinz. „On non-functional requirements“. In: *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*. IEEE. 2007, S. 21–26 (zitiert auf S. 26, 27).

[Gmb18] dc-square GmbH. *MQTT Quality of Service Levels*. 2018. URL: https://www.hivemq.com/ (zitiert auf S. 32).

[HBS+16] P. Hirmer, U. Breitenbücher, A. C. F. da Silva, K. Képes, B. Mitschang, M. Wieland. „Automating the Provisioning and Configuration of Devices in the Internet of Things.“ In: *CSIMQ* 9 (2016), S. 28–43 (zitiert auf S. 21).

[HHM+17] M. Hüffmeyer, P. Hirmer, B. Mitschang, U. Schreier, M. Wieland. „SitAC–a system for situation-aware access control-controlling access to sensor data“. In: (2017) (zitiert auf S. 15).

[Hir18] P. Hirmer. „Anforderungsbasierte Modellierung und Ausführung von Datenflussmodellen“. In: (2018) (zitiert auf S. 40).

[HWS+17] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, S. G. Sáez, F. Leymann. „Situation recognition and handling based on executing situation templates and situation-aware workflows“. In: *Computing* 99.2 (2017), S. 163–181 (zitiert auf S. 15).

[Int18] M. Integrated. *DS18B20*. https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf. 2018 (zitiert auf S. 32).

[KHLL18] A. E. Khaled, A. Helal, W. Lindquist, C. Lee. „IoT-DDL–Device Description Language for the “T” in IoT“. In: *IEEE Access* 6 (2018), S. 24048–24063 (zitiert auf S. 25).

[KHW+17] L. Kassner, P. Hirmer, M. Wieland, F. Steimle, J. Königsberger, B. Mitschang. „The social factory: connecting people, machines and data in manufacturing for context-aware exception escalation“. In: *Proceedings of the 50th Hawaii International Conference on System Sciences*. 2017 (zitiert auf S. 15).

[LL15] I. Lee, K. Lee. „The Internet of Things (IoT): Applications, investments, and challenges for enterprises“. In: *Business Horizons* 58.4 (2015), S. 431–440 (zitiert auf S. 15).

[MAP+18] S. Mubeen, S. A. Asadollah, A. V. Papadopoulos, M. Ashjaei, H. Pei-Breivold, M. Behnam. „Management of service level agreements for cloud services in iot: A systematic mapping study“. In: *IEEE Access* 6 (2018), S. 30184–30207 (zitiert auf S. 31).

[Mon18] MongoDB. *comparison mongoDB*. 2018. URL: https://docs.mongodb.com/manual/reference/operator/query-comparison/ (zitiert auf S. 60).

[Ser18] A. W. Services. *AWS IoT Policy*. 2018. URL: https://aws.amazon.com/de/iot-core/pricing/ (zitiert auf S. 30).

[TD95] R. H. Thayer, M. Dorfman. *System and software requirements engineering*. IEEE Computer Society Press, 1995 (zitiert auf S. 16, 24).

[W3C18] W3C. *Web of Things (WoT) Thing Description*. 2018. URL: https://w3c.github.io/wot-thing-description/ (zitiert auf S. 25).

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more*. Prentice Hall PTR, 2005 (zitiert auf S. 15, 24, 35, 47).

[WMS12] S. Wahle, T. Magedanz, F. Schulze. „The OpenMTC framework—M2M solutions for smart cities and the internet of things“. In: *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*. IEEE. 2012, S. 1–3 (zitiert auf S. 15).

[ZBC+14] A. Zanella, N. Bui, A. Castellani, L. Vangelista, M. Zorzi. „Internet of things for smart cities“. In: *IEEE Internet of Things journal* 1.1 (2014), S. 22–32 (zitiert auf S. 15).

[ZCW+14] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, S. Shieh. „IoT security: ongoing challenges and research opportunities“. In: *Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on*. IEEE. 2014, S. 230–234 (zitiert auf S. 26).

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift