Institut für industrielle Informationstechnik
Institute Industrial IT

Hochschule Hannover
Fakultät I – Elektro- und Informationstechnik

Beschreibung und Bewertung von Middleware-Lösungen

Interne Dokumentation der Forschung aus Arbeitspaket 1 des Projekts IT_SIVA

Gefördert durch:

Bundesministerium für Wirtschaft und Energie

aufgrund eines Beschlusses des Deutschen Bundestages

Industrielle Gemeinschaftsforschung

IT_SIVA

Das IGF-Vorhaben IT_SIVA (19117 N) der Forschungsvereinigung Elektrotechnik beim ZVEI e.V., Lyoner Straße 9, 60528 Frankfurt am Main wird über die AiF im Rahmen des Programms zur Förderung der Industriellen Gemeinschaftsforschung (IGF) vom Bundesministerium für Wirtschaft und Energie aufgrund eines Beschlusses des Deutschen Bundestages gefördert.

Autoren: Prof. Dr. Stefan Heiss
Gajasri Karthikeyan
Maxim Friesen
Prof. Dr. Karl-Heinz Niemann
Kai Steinke
Sebastian Tebbje

I. Inhaltsverzeichnis

I. Inhaltsverzeichnis I
II. Abbildungsverzeichnis VI
III. Definitionen VII
III.1 Architektur VII
III.1.1 SOA - Serviceorientierte Architektur VII
III.1.2 ROA - Ressourcenorientierte Architektur VII
III.1.3 OOA - Objektorientierte Architektur VII
III.1.4 EDA - Ereignisgesteuerte Architektur VII
III.1.5 MOM - Nachrichtenorientierte Middleware VII
III.2 Messaging-Paradigma VIII
III.2.1 Request/Response (deut.: Anfrage/Antwort) VIII
III.2.2 Publish/Subscribe (deut.: Veröffentlichen/Abonnieren) VIII
III.2.3 Point-to-Point (deut.: Punkt-zu-Punkt) VIII
III.3 Echtzeitfähigkeit VIII
III.3.1 Harte Echtzeit VIII
III.3.2 Weiche Echtzeit VIII
III.3.3 Keine Echtzeit IX
III.3.4 Referenzen IX
III.3.5 Übersicht der betrachteten Protokolle IX
1 Constrained Application Protocol (CoAP) 1
1.1 Allgemeine Beschreibung 1
1.2 Ebene 1
1.3 Standardisierung 1
1.4 Architektur 1
1.5 Messaging-Paradigma 2
1.6 Echtzeitfähigkeit 2
1.7 Servicequalität 2
1.8 Sicherheit 2
1.9 Physikalische Anforderungen 2
1.10 Verfügbare Implementierungen 2
1.11 Referenzen 2
2 IoTivity 3
2.1 Allgemeine Beschreibung 3
2.2 Ebene 5

2.3 Standardisierung 5
2.4 Architektur 6
2.5 Messaging-Paradigma 6
2.6 Echtzeitfähigkeit 6
2.7 Servicequalität 6
2.8 Sicherheit 6
2.9 Physikalische Anforderungen 7
2.10 Verfügbare Implementierungen 7
2.11 Referenzen 8
3 LWM2M 9
3.1 Allgemeine Beschreibung 9
3.2 Ebene 10
3.3 Standardisierung 10
3.4 Architektur 11
3.5 Messaging-Paradigma 11
3.6 Echtzeitfähigkeit 12
3.7 Servicequalität 12
3.8 Sicherheit 12
3.9 Physikalische Anforderungen 12
3.10 Verfügbare Implementierungen 12
3.11 Referenzen 12
4 AllJoyn 13
4.1 Allgemeine Beschreibung 13
4.2 Ebene 15
4.3 Standardisierung 15
4.4 Architektur 15
4.5 Messaging-Paradigma 15
4.6 Echtzeitfähigkeit 15
4.7 Servicequalität 15
4.8 Sicherheit 16
4.9 Physikalische Anforderungen 16
4.10 Verfügbare Implementierungen 17
4.11 Referenzen 17
5 Open Platform Communication Unified Architecture (OPC UA) 18
5.1 Allgemeine Beschreibung 18
5.2 Ebene 18

5.3 Standardisierung 18
5.4 Architektur 18
5.5 Messaging-Paradigma 19
5.6 Echtzeitfähigkeit 19
5.7 Servicequalität 19
5.8 Sicherheit 19
5.9 Physikalische Anforderungen 21
5.10 Verfügbare Implementierungen 21
5.11 Referenzen 22
6 Streaming Text Oriented Messaging Protokoll (STOMP) 23
6.1 Allgemeine Beschreibung 23
6.2 Ebene 23
6.3 Standardisierung 23
6.4 Architektur 24
6.5 Messaging-Paradigma 24
6.6 Echtzeitfähigkeit 25
6.7 Servicequalität 25
6.8 Sicherheit 26
6.9 Physikalische Anforderungen 26
6.10 Verfügbare Implementierungen 26
6.11 Referenzen 26
7 Java Message Service (JMS) 27
7.1 Allgemeine Beschreibung 27
7.2 Ebene 27
7.3 Standardisierung 27
7.4 Architektur 27
7.5 Messaging-Paradigma 28
7.6 Echtzeitfähigkeit 29
7.7 Servicequalität 29
7.8 Sicherheit 30
7.9 Physikalische Anforderungen 30
7.10 Verfügbare Implementierungen 30
7.11 Referenzen 30
8 Message Queuing Telemetry Transport (MQTT) 31
8.1 Allgemeine Beschreibung 31
8.2 Ebene 31

8.3 Standardisierung 31
8.4 Architektur 31
8.5 Messaging-Paradigma 31
8.6 Echtzeitfähigkeit 32
8.7 Servicequalität 32
8.8 Sicherheit 32
8.9 Physikalische Anforderungen 33
8.10 Verfügbare Implementierungen 33
8.11 Referenzen 33
9 Data Distribution Service (DDS) 34
9.1 Allgemeine Beschreibung 34
9.2 Ebene 35
9.3 Standardisierung 35
9.4 Architektur 35
9.5 Messaging-Paradigma 36
9.6 Echtzeitfähigkeit 36
9.7 Servicequalität 36
9.8 Sicherheit 36
9.9 Physikalische Anforderungen 37
9.10 Verfügbare Implementierungen 37
9.11 Referenzen 37
10 Advanced Message Queuing Protocol (AMQP) 38
10.1 Allgemeine Beschreibung 38
10.2 Ebene 38
10.3 Standardisierung 38
10.4 Architektur 38
10.5 Messaging-Paradigma 39
10.6 Echtzeitfähigkeit 39
10.7 Servicequalität 39
10.8 Sicherheit 40
10.9 Physikalische Anforderungen 40
10.10 Verfügbare Implementierungen 40
10.11 Referenzen 40
11 Web Services 41
11.1 Allgemeine Beschreibung 41
11.2 Ebene 42

11.3 Standardisierung 43
11.4 Architektur 43
11.5 Messaging-Paradigma 43
11.6 Echtzeitfähigkeit 43
11.7 Servicequalität 44
11.8 Sicherheit 44
11.9 Physikalische Anforderungen 44
11.10 Verfügbare Implementierungen 44
11.11 Referenzen 45

II. Abbildungsverzeichnis

Abbildung 1 Harte vs. weiche Echtzeitanwendungen[1]. VIII
Abbildung 2 Grundlegendes Server-/Client-Anforderungs-/Antwortmodell 2
Abbildung 3 Die OCF-Framework-Architektur[1]. 3
Abbildung 4 Die Funktionsblöcke des OCF-Frameworks[8]. 4
Abbildung 5 Das Konzept und die Darstellung im OCF-Ressourcenmodell[8]. 5
Abbildung 6 Eine Beispielsammlung, die das Konzept der Links im OCF-Ressourcenmodell veranschaulicht[8]. 5
Abbildung 7 SRM- und PM-Schicht des OCF[10]. 6
Abbildung 8 Gerätestapel der IoTivity-Implementierung für uneingeschränkte und eingeschränkte Geräte[9]. 7
Abbildung 9 LWM2M Request/Response SMS Frame (ohne Token)[11]. 9
Abbildung 10 Der Protokollstapel des LWM2M Enabler[11]. 9
Abbildung 11 Das LWM2M-Datenmodell[12] 11
Abbildung 12 Die LWM2M-Architektur[13] 11
Abbildung 13 AllJoyn-Busse von zwei verschiedenen Vorrichtungen, die verschiedene Prozesse verbinden, die als Hexagone dargestellt sind. Die Prozesse werden über ihre Verbindungsnamen angesprochen und sind in dieser Abbildung vereinfacht (z.B. :1.2)[10]. 14
Abbildung 14 Verbundene AllJoyn-Busse von zwei physikalisch getrennten Geräten[10]. 14
Abbildung 15 AllJoyn Protokollstapel[2]. 14
Abbildung 16 Fernzugriff auf die AllJoyn-Netzwerkarchitektur[2]. 14
Abbildung 17 AllJoyn Sicherheitsarchitektur[11] 16
Abbildung 18 AllJoyn Distributed Bus mit Thin-Core-Bibliotheken[9]. 16
Abbildung 19 Optionen des OPC UA-Protokolls[2]. 18
Abbildung 20 OPC UA Sicherheitsarchitektur[11] 20
Abbildung 21 Einfache Public-Key-Infrastruktur 21
Abbildung 22 Message-Broker-Lösung[3]. 23
Abbildung 23 STOMP Datenrahmenstruktur [1] 24
Abbildung 24 SEND-Rahmen[1] 24
Abbildung 25 Heart Beat Header[1]. 25
Abbildung 26 Punkt-zu-Punkt-Messaging[2]. 28
Abbildung 27 Veröffentlichen/Abonnieren von Messaging[2]. 28
Abbildung 28 JMS-Verwaltung[1]. 29
Abbildung 29 MQTT Veröffentlichungs-/Abonnementarchitektur 32
Abbildung 30 Festes Header-Format[1] 33
Abbildung 31 Struktur eines MQTT Control Packets[1]. 33
Abbildung 32 DDS-Architektur in einem Edge-Netzwerk mit Edge-Geräten 34
Abbildung 33 Eine DDS-Domäne im globalen Datenraum 35
Abbildung 34 AMQP-Nachrichtenparadigma[9]. 39
Abbildung 35 Allgemeines SOAP/WSDL-Modell mit einem Client und einem Server mit verschiedenen Technologien 41
Abbildung 36 Allgemeines REST-Modell mit einem Client und einem Server mit verschiedenen Technologien 42

III. Definitionen

III.1 Architektur

III.1.1 SOA - Serviceorientierte Architektur

In der SOA kombiniert oder komponiert eine Anwendung ihre dem Benutzer präsentierte Funktionalität auf der Grundlage verschiedener extern abgefragter Dienste. Informationen aus den Diensten werden gesammelt, indem man sie mit den erforderlichen Parametern versorgt und die Ergebnisse für die Benutzer kombiniert. Die Verarbeitung erfolgt überwiegend durch die externen Dienste. Als Beispiel soll ein WS Informationen darüber liefern, ob und zu welchem Zeitpunkt die Internationale Raumstation an einem bestimmten Ort auf der Welt sichtbar ist. Mit einer SOA würde die Anwendung externe Dienste mit den Parametern „Stadt“ und „Datum“ abfragen, um, basierend auf dem Standort und den lokalen Wetterberichten, eine Wahrscheinlichkeit für die Sichtbarkeit der ISS zu ermitteln.

III.1.2 ROA - Ressourcenorientierte Architektur

In ROA übernehmen Anwendungen den größten Teil der Verarbeitung selbst, verwenden aber externe Ressourcen als Dateneingabe. Ressourcen sind Softwarekomponenten wie Datenstrukturen oder diskreter Code. Informationen werden als statische Ressourcen von externen Dienstleistern abgerufen und die Kalkulation intern durchgeführt. Im Vergleich zum obigen ISS-Dienst würde die Anwendung nun eine Ressource abrufen, die den ISS-Orbit während des angeforderten Zeitraums repräsentiert, und eine Ressource, die die Wettervorhersage für die Stadt enthält. Die Wahrscheinlichkeit, die ISS zu sehen, wird dann berechnet und dem Benutzer anhand der erfassten Daten präsentiert.

III.1.3 OOA - Objektorientierte Architektur

In OOA erfolgt die Kommunikation zwischen verschiedenen Entitäten auf einer sehr niedrigen Abstraktionsschicht. Objekte enthalten Methoden und Eigenschaften sowie Referenzen auf andere entfernte Objekte, deren Methoden aufgerufen werden können. Alle Objekte haben den Zustand, in dem sie ermittelt und aufeinander bezogen werden müssen, um eine spätere Interaktion zu ermöglichen. Um Informationen über die Temperatur eines CPU-Kerns zu sammeln, greift eine Anwendung in einer OOA auf einen bestimmten Adressraum innerhalb der Vorrichtung zu, die die CPU mit allen relevanten Informationen enthält, indem sie die erforderliche Methode in einem entfernten Objekt aufruft.

III.1.4 EDA - Ereignisgesteuerte Architektur

EDA kann in einem dezentralen System realisiert werden, in dem jede Entität ihre eigene Datenbank hat. Änderungen in der Datenbank sind Ereignisse, die weitere Ereignisse bei anderen Entitäten auslösen können. Alle Ereignisse, die für die Auslösung weiterer Prozesse relevant sein könnten, können abonniert werden. Eine Entität signalisiert allen interessierten Parteien, wenn ein solches Ereignis eintritt. Wenn z.B. in einem Gefrierschrank die Temperatur erfasst wird und einen bestimmten Wert überschreitet, wird ein Ereignis ausgelöst, wodurch die Kühlgeräte ihre Leistung und die Kühlung erhöhen.

III.1.5 MOM - Nachrichtenorientierte Middleware

Im Gegensatz zu High-Level-Konzepten wie SOA oder ROA ist eine MOM eine eigentliche Implementierung einer Middleware-Plattform. Es kann zur Implementierung einer SOA, einer EDA oder anderer Architekturen verwendet werden. Normalerweise ermöglicht eine MOM asynchronem´s Messaging zwischen verschiedenen Anwendungen, bei dem ein MOM-Server die Nachrichten speichert und weiterleitet. Beispielsweise kann ein Webservice Kundenanfragen bearbeiten, indem er sie an eine Backend-Anwendung weiterleitet. Ein MOM zwischen dem Webservice und der Backend-Anwendung würde deren lose Kopplung ermöglichen und die Integration von Funktionen wie einem

Warteschlangensystem ermöglichen. Die lose Kopplung wird verwendet, um das Risiko zu verringern, dass Änderungen, die innerhalb eines Elements vorgenommen werden, zu unerwarteten Änderungen innerhalb anderer Entitäten führen. Es ermöglicht eine einfache Skalierbarkeit, indem es Abhängigkeiten zwischen den Entitäten minimiert. Die Warteschlange kann wichtig sein, wenn mehr Webservice-Anfragen vorliegen, als das Backend gleichzeitig verarbeiten kann. Ohne ein MOM würde die Anwendung einige Anfragen verwerfen oder eine integrierte Warteschlange benötigen. Ein MOM kann solche Funktionen bereits integrieren, ohne den Webservice oder die Anwendung ändern zu müssen.

III.2 Messaging-Paradigma

III.2.1 Request/Response (deut.: Anfrage/Antwort)

Request/Response ist eine bidirektionale Kommunikation über einen Kanal zwischen einem Client und einem Server. Der Client sendet eine Datenanfrage und der Server empfängt und verarbeitet die Anfrage und sendet eine Antwort.

III.2.2 Publish/Subscribe (deut.: Veröffentlichen/Abonnieren)

Publish/Subscribe ist ein Nachrichtenmuster, bei dem sich die Publisher, die Nachrichten senden, und die Abonnenten, die Nachrichten empfangen, nicht kennen, was bedeutet, dass sie nur wissen, welche Art von Nachricht gesendet wird, aber nicht von wem oder an wen. Es kann mehrere Publisher und/oder Abonnenten geben.

III.2.3 Point-to-Point (deut.: Punkt-zu-Punkt)

Eine Point-to-Point-Verbindung ist eine Direktverbindung zwischen zwei Netzwerkkomponenten. Dabei kann die Verbindungsart fest geschaltet oder über eine Vermittlungsstelle (Broker) erfoglen. Nachrichten werden innerhalb der Warteschlange gehalten, bis sie vom Empfänger abgerufen werden. Über die Nachrichtenwarteschlange können verschiedene Quality-of-Service Optionen durchgesetzt werden, da der Broker allein entscheidet, wann eine Nachricht erfolgreich zugestellt wurde, ohne dass der Absender weitere Aufmerksamkeit benötigt.

III.3 Echtzeitfähigkeit

III.3.1 Harte Echtzeit

Harte Echtzeitanwendungen können keine Nachrichtenzustellfristen verpassen (jeder verpasste Frist gilt als Systemausfall). Der Datentransport ist daher völlig deterministisch.

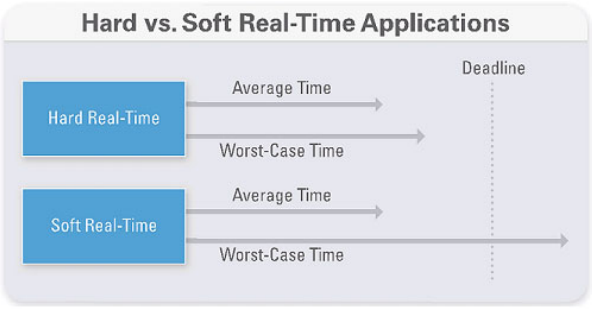

Abbildung 1 Harte vs. weiche Echtzeitanwendungen [1].

III.3.2 Weiche Echtzeit

Weiche-Echtzeit-Anwendungen tolerieren einige verpasste Fristen, wenn die durchschnittliche Zustellung aller Nachrichten diese Frist nicht überschreitet.

Sicherheitskritische Anwendungen in Kernkraftwerken oder Raumfahrtprojekten sind beispielsweise harte Echtzeitanwendungen. Multimedia-Anwendungen wie Videoanrufe oder Webchat-

Anwendungen können als Weiche-Echtzeit-Anwendungen betrachtet werden, bei denen erhebliche Verzögerungen zulässig sind.

III.3.3 Keine Echtzeit

In nicht-Echtzeitfähigen Anwendung ist der Datentransport nicht deterministisch.

III.3.4 Referenzen

[1] "Echtzeitsystem." [Online]. Verfügbar unter: http://www.ni.com/white-paper/14238/de/. [Zugriff: 29. Januar 2018].

III.3.5 Übersicht der betrachteten Protokolle

Tabelle 1: Übersicht der Protokolle

Protokoll		Organisation	Protokolltyp	Transport-protokoll	Open-Source	Echtzeit	Zusätzliche QoS	Beschränkte Systeme	Nachrichten-paradigma	Sicherheit		
										E2E-Verschl.	Authent.	Zugriffs-Kontrolle
WS	SOAP	W3C	MP	TCP,UDP	✓	✗	✓	✗	RR	TLS,WS-Sec.	WS-Trust	✓
	REST	W3C	MP	TCP,UDP	✓	✗	✗	✗	RR	TLS, DTLS	✗	✗
CoAP		IETF	MP	REST/UDP	✓	✗	○	✓	RR	DTLS	✗	✗
MQTT		OASIS	MP	TCP	✓	○	✓	✓	PS	TLS	TLS	✓
AMQP		OASIS	MP	TCP	✓	✗	✓	✗	RR,PS, PP	TLS	SASL	✓
XMPP		IETF	MP	TCP	✓	○	✗	✗	RR,PS	TLS	SASL	✓
IoTivity		OCF	FP	CoAP/UDP	✓	✗	○	✓	RR	DTLS	P	✓
LWM2M		OMA	FP	CoAP/UDP	✓	✗	○	✓	RR	DTLS	P	✓
OPC UA		OPCF	FP	SOAP/TCP, UDP	✓	✗	○	✓	RR,PS	TLS,P, WS-Sec.	P	✓
DDS		OMG	FP	TCP, UDP	✓	✓	✓	✓	PS	P	P	✓

Tabelle 2: Einsatzgebiete der Protokolle

Protokoll	Einsatzgebiet
WS SOAP	Remote Procedure Calls
WS REST	Remote Procedure Calls
COAP	Resourcenbeschränkte Geräte
MQTT	Smart Home, Resourcenbeschränkte Geräte
AMQP	Finanzsektor
XMPP	Instant Messaging
IoTivity	Smart Home
LWM2M	Machine-to-Machine
OPC UA	Machine-to-Machine
DDS	Machine-to-Machine

1 Constrained Application Protocol (CoAP)

1.1 Allgemeine Beschreibung

Das Constrained Application Protocol (CoAP) ist ein Web-Transfer-Protokoll, das als Light-HTTP bezeichnet werden kann, da es mehrere Ähnlichkeiten zwischen diesen Protokollen gibt. CoAP ist ein Anwendungsschichtprotokoll, das auf der Architektur des Representational State Transfer (REST) basiert [8]. Es ist für Knoten und Geräte mit Einschränkungen bei Strom-, Speicher- und Verarbeitungsressourcen konzipiert und ermöglicht die Interaktion zwischen Clients und Servern über das Internet. CoAP wurde entwickelt, um mit begrenzter Rechenleistung und begrenztem Speicher zu kommunizieren, der typischerweise in Umgebungen mit niedriger Bitrate arbeitet. Es verwendet einen minimierten HTTP-Header und folgt dem Request-/Response-Modell. Ressourcen, die tatsächlich zugängliche Daten sind, werden über Universal Resource Identifiers (URI) identifiziert und angesprochen. Anfragen und Antworten werden asynchron über CoAP-Nachrichten ausgetauscht. Durch die Verwendung einer erweiterten GET-Methode mit der Observe-Erweiterung [5] unterstützt CoAP auch einen publish/subscribe-ähnlichen Ansatz. Es ermöglicht das periodische Abrufen einer Ressource ohne die Komplexität und den großen Aufwand, die durch das regelmäßige HTTP-Polling verursacht werden. Das Interaktionsmodell von CoAP ist wie das Client/Server-Modell von HTTP. Der Client sendet eine Anforderung über die URI, um eine Ressource abzurufen oder zu manipulieren, und der Server sendet eine Darstellung der abgerufenen oder geänderten Ressource. Bei Verwendung der Observe-Erweiterung können bestimmte Ressourcen abonniert werden. Der Server sendet regelmäßig die aktuelle Ressourcendarstellung an den Teilnehmer. CoAP hat nur einen 4-Byte-Header mit UDP als Standard-Transportprotokoll. Die Kommunikation mit UDP ist nicht zuverlässig, so dass CoAP selbst zuverlässige Mechanismen bereitstellt. CoAP hat einen geringen Overhead und bietet keine differenzierte QoS. Um einen Peer-to-Peer-Stil der Kommunikation zu unterstützen, kann CoAP One-to-Many-Funktionen über den Einsatz von IP-Multicast unterstützen. CoAP bietet außerdem Erkennungsmechanismen, um Details zu Ressourcen aus URIs [4] abzurufen.

Ähnlich wie bei HTTP gibt es vier verschiedene Methoden, die im CoAP-Standard [1] definiert sind:

- GET - für das Abrufen von Ressourcen
- POST - für die Übermittlung von Informationen
- PUT - für die Aktualisierung der Ressource auf dem Server
- DELETE - zum Löschen von Ressourcen

1.2 Ebene

Es ist ein Application Layer Protokoll und wurde entwickelt, um mit HTTP und dem RESTful Web über einfache Proxies zu interagieren. Dies ermöglicht die Integration in bestehende RESTful-Architekturen und eine allgemein hohe Interoperabilität zwischen verschiedenen Technologien.

1.3 Standardisierung

CoAP wurde in der IETF als RFC 7252 [3] standardisiert.

1.4 Architektur

CoAP basiert auf REST und ist damit eine ressourcenorientierte Architektur, die zu einer serviceorientierten Architektur ausgebaut werden kann.

1.5 Messaging-Paradigma

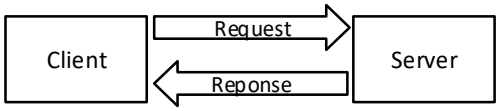

Abbildung 2 Grundlegendes Server-/Client-Request-/Response-modell

CoAP arbeitet normalerweise mit dem Request-/Response-modell, wie inAbbildung 2. dargestellt.

1.6 Echtzeitfähigkeit

Mit CoAP ist keine harte Echtzeitkommunikation möglich, da es auf der Applikationsebene arbeitet und auf nativer UDP-Kommunikation basiert. CoAP verwendet das klassische Request/Response-Modell, das für die Echtzeit-Kommunikation keinen optimierten Ansatz darstellt, da die Art der Round-Trip-Kommunikation zusätzliche Verzögerungen verursacht. Es gibt jedoch Versuche, eine vollständig deterministische CoAP-Kommunikation zu erreichen [6], aber dies sind experimentelle Erweiterungen und gehören nicht zur offiziellen Normspezifikation.

1.7 Servicequalität

CoAP hat die QoS-Fähigkeiten eingeschränkt. Es verwendet Nachrichten-IDs, um nach Duplikaten zu suchen, und bietet Nachrichten, die vom Empfänger bestätigt werden müssen. Die IDs werden verwendet, um Nachrichten und deren Bestätigung zu verfolgen. Dies bietet eine sehr einfache verbindungsorientierte QoS.

1.8 Sicherheit

Das Constrained Application Protocol läuft über UDP [1] und verwendet Datagram Transport Layer Security (DTLS) [9]. CoAP-Methoden entsprechen am weitesten der Funktion und Bedeutung von HTTP-Methoden. DTLS bietet Authentifizierung, Autorisierung, Datenintegrität und Vertraulichkeit der Datenherkunft. DTLS besteht aus zwei Phasen: Ein Handshake zwischen den beiden kommunizierenden Maschinen, bei dem sich beide authentifizieren und den anderen mit Zertifikaten validieren müssen. Die zweite Phase ist die verschlüsselte Datenübertragungsphase, die authentifizierte und verschlüsselte Daten unter Verwendung der in der Handshake-Phase erstellten Sicherheitsschlüssel sendet [10]. Die Verwendung der standardmäßigen DTLS-Chiffriersuiten ist für ressourcenbeschränkte Geräte möglicherweise nicht effizient, da der Overhead und die Verwendung von Zertifikaten die Leistung bei geringer Leistung beeinträchtigen [2]. Es muss sorgfältig abgewogen werden, wobei die spezifischen Verschlüsselungs-Suiten zu berücksichtigen sind, ob sie in der verwendeten CoAP-Anwendung anwendbar sind.

1.9 Physikalische Anforderungen

Das Constrained Application Protocol zeichnet sich durch einen geringen Header-Overhead und eine hohe Parsing-Komplexität aus. CoAP arbeitet ressourcenarm. Es handelt sich um ein Protokoll mit geringer Leistung und geringem Platzbedarf, das für eingeschränkte Geräte entwickelt wurde.

1.10 Verfügbare Implementierungen

CoAP ist ein offener Standard. Es gibt verschiedene plattformunabhängige Implementierungen für die meisten relevanten Programmiersprachen [7].

1.11 Referenzen

1] CoAP, RFC 7252 Constrained Application Protocol, [Online]. Verfügbar: http://coap.technology/[Zugriff: 10-Sep-2017].

Das Constrained Application Protocol (CoAP), IETF, [Online]. Verfügbar unter: https://tools.ietf.org/html/rfc7252[Zugriff: 12-Sep-2017].

3] libcoap: "C-Implementierung von CoAP"[Online]. Verfügbar: https://libcoap.net/ [Zugriff: 12-Sep-2017].

Carsten Bormann, Angelo P. Castellani, Zack Shelby, "CoAP. Ein Anwendungsprotokoll für Milliarden winziger Internetknoten", 2012.

[5] K. Hartke, "Observing Resources in the Constrained Application Protocol (CoAP)", Sep. 2015.

Björn Konieczek, Michael Rethfeldt, Frank Golatowski, Dirk Timmermann, "A Distributed Time Server for the Real-Time Extension of CoAP", 2016.

[7] CoAP-Technologie, Implementierungen, [Online] verfügbar: http://coap.technology/impls.html. [Zugriff: 04-Sep-2017].

[8] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", University of California, Irvine, 2000.

[9] T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) - Profile für das Internet der Dinge", RFC 7925, 2016.

Ho-won Kim, Dooho Choi, "Informationssicherheitsanwendungen: 16. Internationaler Workshop, WISA, 2015.

2 IoTivity

2.1 Allgemeine Beschreibung

IoTivity ist ein Open-Source-Software-Framework, das verschiedene Standards der Open Connectivity Foundation (OCF) implementiert [1]. Das OCF war früher als Open Interconnect Consortium (OIC) bekannt. Das OCF ist eine Standardorganisation, die sowohl Spezifikationen, als auch Interoperabilitätsrichtlinien entwickelt und Geräte für das Internet der Dinge (IoT) zertifiziert. IoTivity zielt darauf ab, einen Rahmen für Kommunikation und Interoperabilität zu schaffen, der an die verschiedenen IoT-Märkte wie Industrie, Gesundheitswesen und Verbraucher angepasst werden kann.

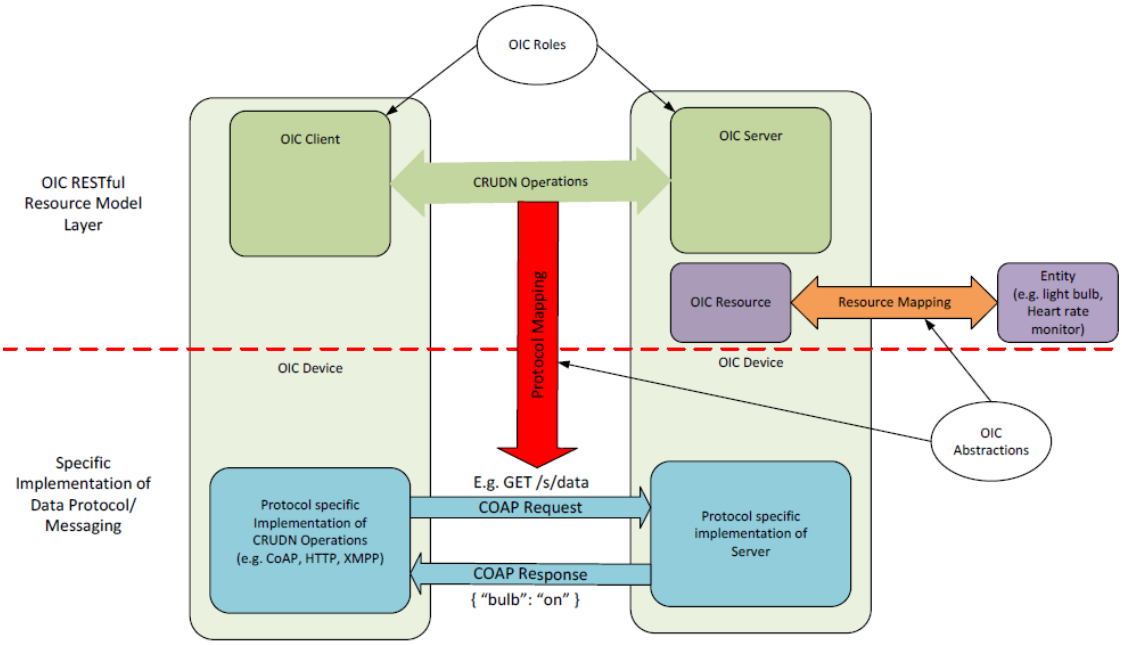

Abbildung 3 Die OCF-Framework-Architektur [1]

Die allgemeine Architektur, wie inAbbildung 3 dargestellt, basiert auf den Prinzipien des Representational State Transfer (REST) und verwendet das Constrained Application Protocol (CoAP) über UDP als Basis-Kommunikationsprotokoll. REST ist ein Architekturstil zur Verwaltung von Zustandsinformationen in vernetzten Systemen. Es behandelt Objekte (jede Art von Daten oder Informationen) auf Servern als Ressourcen und stellt diese über einen Uniform Resource Identifier (URI) bereit. Beim Zugriff werden die Darstellungen der Ressourcen abgerufen. Diese Darstellungen sind zustandslos, da die Kommunikation zwischen Server und Client immer alle Informationen enthält, die zur Ausführung der Anforderung erforderlich sind. CoAP funktioniert wie HTTP, ist aber speziell für eingeschränkte Geräte konzipiert, z.B. Geräte mit Low-Power-Mikrocontrollern, kleinen Mengen an RAM und Flash und begrenzter Bandbreite.

Clients und Server kommunizieren auf der Grundlage der einfachen Request-/Response-mechanismen mit CRUDN-Befehlen (Create, Read Update, Delete, Notify). Diese werden in der protokollspezifischen Implementierung, die auf dem Gerät läuft, auf die CoAP-Operationen abgebildet. Server stellen Ressourcen bereit, die Einheiten wie Sensoren, Aktoren usw. zugeordnet sind. (z.B. Glühbirne, Pulsmesser). Ressourcen können von Clients abgerufen oder manipuliert werden, um

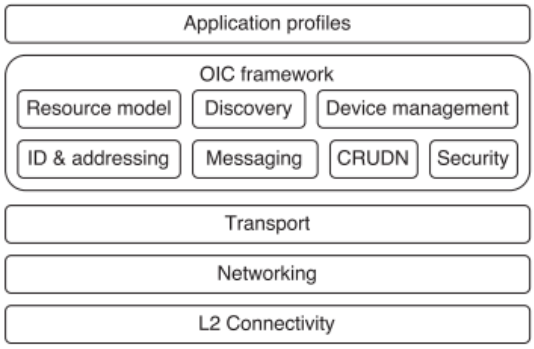

Abbildung 4 Die Funktionsblöcke des OCF-Frameworks [8]

Statusinformationen abzurufen oder das Gerät und seine einzelnen Einheiten zu steuern.

Wie in Abbildung 4 zu sehen ist, ist das OCF-Framework zwar oberhalb der Transportschicht aufgebaut, erlaubt aber unterschiedliche Anwendungsprofile auf der eigenen Schicht. Dies kann unterschiedliche Anwendungsbereiche (z.B. Verbraucher, Unternehmen, Industrie, etc.) umfassen. Das OCF-Framework spezifiziert mehrere Kernfunktionalitäten (Abbildung 3), die Teil der OCF-Clients und -Server sind. Die Kernfunktionsblöcke sind:

- **ID & Adressierung:** Definiert, wie Elemente eindeutig durch Identifikatoren benannt werden, um eine einheitliche Adressierung über Kontexte und Domänen hinweg zu ermöglichen. Uniform Resource Identifiers (URI's) sind ein Beispiel dafür, wie Ressourcen aufgerufen werden können.
- **Messaging**: CRUDN-Operationen werden für das Daten- und Ressourcenmanagement verwendet. Diese Operationen werden einem bestimmten Kommunikationsprotokoll wie CoAP zugeordnet, indem die Methoden Get, Post, Put und Delete verwendet werden.
- **Ressourcenmodell**: Definiert das interoperable semantische Modell, das es Clients ermöglicht, mit Servern und deren Entitäten zu interagieren. Die Abbildung 5 zeigt die Konzepte und die Darstellung des Ressourcenmodells. Eine Ressource bezieht sich auf eine Entität, die etwas Sichtbares, Zugängliches und Manipulierbares im Netzwerk beschreibt. Der Ressourcentyp wird mit seinen Eigenschaften beschrieben und kann ein Sensor, eine Batterie, ein Schalter usw. sein. Die URI wird verwendet, um die Ressource anzusprechen und darauf zuzugreifen.

URIs aus anderen Ressourcen werden verwendet, um Beziehungen zu erstellen und Kollektionen aufzubauen, die Strukturen von Ressourcen für verschiedene Anwendungskontexte miteinander verbinden. Die Abbildung 6 zeigt ein Beispiel, bei dem eine Ressource auf andere Ressourcen im Anwendungskontext eines Luftstromsteuerungssystems verweist. Nach dem REST-Prinzip werden Ressourcen immer als Darstellungen ihres aktuellen Zustands abgerufen.

- **Discovery**: Definiert die Endpunkt- und Ressourcenerkennung. Wann immer eine neue Ressource instanziiert wird, aktualisiert der Server seine lokale Kernressource und fügt sie zu seiner Liste der auffindbaren Ressourcen hinzu. Andere Geräte können diese Liste mit Unicast- oder Multicast-Anfragen abrufen, um sich über die verfügbaren Ressourcen und deren URIs zu informieren.

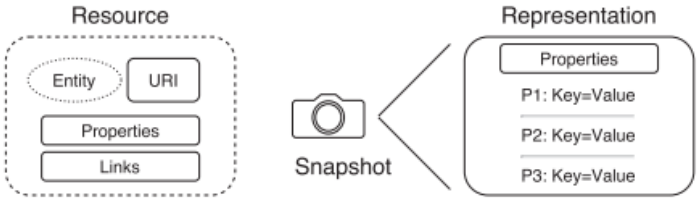

Abbildung 5 Das Konzept und die Darstellung im OCF-Ressourcenmodell [8]

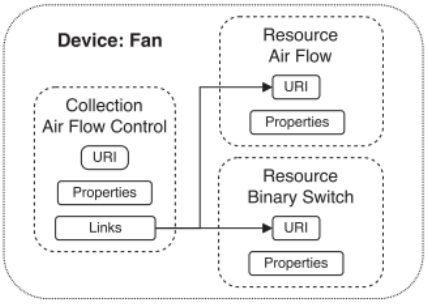

Abbildung 6 Eine Beispielsammlung, die das Konzept der Links im OCF-Ressourcenmodell veranschaulicht [8]

- **Geräteverwaltung**: Ein optionaler Kernressourcentyp namens Wartung ermöglicht es Administratoren, die Geräte aus der Ferne zu verwalten und zu warten. Es kann Eigenschaften wie Werksreset oder Neustart oder andere clientspezifische Wartungs- und Diagnoseeigenschaften enthalten.
- **Sicherheit**: Gibt die Sicherheitsrichtlinien und ein dreistufiges Sicherheitskonzept an. Zunächst wird eine eindeutige DeviceID verwendet, um eine Verbindung zwischen Server und Client herzustellen. Zweitens wird auf dem Kommunikationskanal eine End-to-End-Verschlüsselung aufgebaut. Drittens entscheidet der Secure Resource Manager (SRM) des Servers, ob der Client die Berechtigung zum Zugriff auf die angeforderten Ressourcen hat. Weitere Informationen finden Sie unter Unterpunkt *Sicherheit*.

2.2 Ebene

IoTivity arbeitet auf einer hohen Abstraktionsschicht und nutzt CoAP zur Kommunikation. Die APIs sind so konzipiert, dass sie jedes Anwendungsprofil unterstützen, das von der Gesundheitsversorgung bis hin zu Anwendungsfällen in Unternehmen und Industrie reicht. Die Denkweise hinter IoTivity ist es, die Interoperabilität zwischen IoT-Geräten auf der ganzen Welt unabhängig von deren Anwendungskontexten zu gewährleisten.

2.3 Standardisierung

Das OCF gehört zu den größten Organisationen für industrielle Konnektivitätsstandards für IoT [6], die aus Mitgliedern wie Samsung, Intel und Cisco [7] bestehen. Eine zukünftige Unterstützung in industriellen Anwendungen und eine breite Akzeptanz ist daher sehr wahrscheinlich.

2.4 Architektur

Die OCF-Spezifikation hat eine Resource-Oriented Architecture (ROA) angenommen, was bedeutet, dass Dinge, Informationen und Konzepte als Ressourcen dargestellt werden. Darüber hinaus kann eine Service-Oriented Architecture (SOA) aufgebaut werden, da sich ROA und SOA nicht ausschließen.

2.5 Messaging-Paradigma

IoTivity bildet alle seine Befehle auf CoAP-Request/Response-Messages ab. Daher basiert die Kommunikation auf CRUDN (Create, Retrieve, Update, Delete, Notify) Operationen. Benachrichtigungsoperationen werden durch den Einsatz der CoAP Observer Extension [5] realisiert. Es ermöglicht das periodische Abrufen von Ressourcen ohne die Komplexität und den großen Aufwand, die durch das regelmäßige HTTP-Polling verursacht werden.

2.6 Echtzeitfähigkeit

IoTivity verwendet CoAP als Kommunikationsprotokoll. Da CoAP keine sofort einsatzbereiten Echtzeitfunktionen bietet, ist auch IoTivity betroffen. CoAP verwendet das klassische Request/Response-Modell, das für die RT-Kommunikation keinen optimierten Ansatz darstellt, da die Art der Round-Trip-Kommunikation zusätzliche Verzögerungen verursacht. Es gibt Versuche, eine vollständig deterministische CoAP-Kommunikation zu erreichen [4], aber dies sind experimentelle Erweiterungen und noch nicht Teil der offiziellen Standardspezifikation.

2.7 Servicequalität

Ähnlich wie bei den Echtzeitaspekten ist IoTivity an die QoS-Fähigkeiten von CoAP gebunden. Regular CoAP hat eingeschränkte QoS-Fähigkeiten. Es verwendet Nachrichten-IDs, um nach Duplikaten zu suchen, und bietet Nachrichtentypen, die bestätigt werden müssen. Die IDs werden verwendet, um Nachrichten und deren Bestätigung zu verfolgen. Dies bietet eine sehr einfache verbindungsorientierte QoS.

2.8 Sicherheit

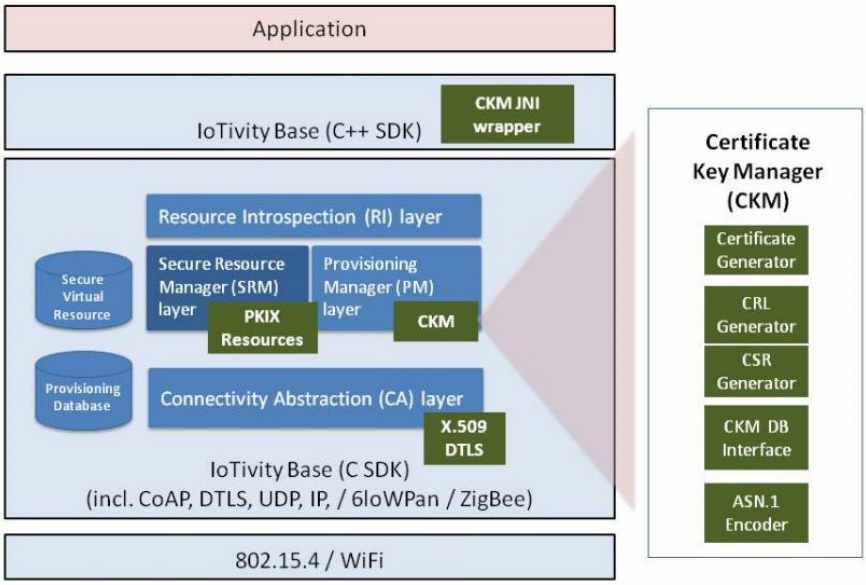

Abbildung 7 SRM- und PM-Schicht des OCF [10].

IoTivity nutzt DTLS, um sichere End-to-End-Kommunikationskanäle zwischen Servern und Clients zu schaffen. Ein Secure Resource Manager (SRM) und ein Provisioning Manager (PM) sind Teil jedes

IoTivity-Gerätestacks. Die Ressourcen-Introspektionsschicht ist für die Analyse und Interpretation der IoTivity-Nachrichten-Nutzlasten verantwortlich, sobald sie von den sicheren Schichten genehmigt wurden (z.B. eine Ressourcenzugriffsanforderung). Die Connectivity-Abstraction (CA)-Schicht enthält die CoAP-Funktionalitäten und verschiedene Transportadapter (IP, BT etc.).

Das PM registriert zunächst das Gerät und konfiguriert die Zugangskontrolle. Es legt die Anmeldeinformationen und Zugriffsrichtlinien für die gegenseitige Authentifizierung fest und hält den Certificate Key Manager (CKM).

Das SRM verwaltet Secure Virtual Resources (SVR) wie Access Control Lists (ACL) und Anmeldeinformationen, die vom PM bereitgestellt werden. Seine Policy-Engine gewährt oder verweigert Ressourcenzugriffsanforderungen basierend auf der ACL, der Anforderer-ID, dem Gerätestatus usw. SVR's werden in der Provisionierungsdatenbank gespeichert.

2.9 Physikalische Anforderungen

Während uneingeschränkte Geräte die C++- und C-basierten APIs nutzen können und in Zukunft neben CoAP möglicherweise verschiedene Kommunikationsprotokolle nutzen, haben eingeschränkte Geräte die Möglichkeit, den Thin Device Stack zu verwenden, der sich auf die C API und CoAP für die Kommunikation beschränkt (Abbildung 8). Der Thin-Stack optimiert den Leistungs- und Bandbreitenbedarf und ermöglicht den Einsatz auf Embedded-Systemen mit begrenzten Ressourcen.

2.10 Verfügbare Implementierungen

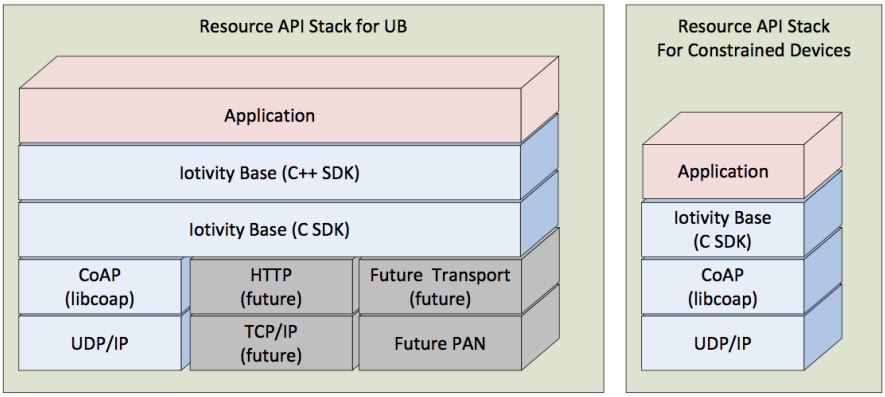

Abbildung 8 Gerätestapel der IoTivity-Implementierung für uneingeschränkte und eingeschränkte Geräte [9].

IoTivity basiert auf den offenen Standards des OCF und wird als C, C++ und Java APIs zur Verfügung gestellt. Es kann auf verschiedenen Betriebssystemen eingesetzt werden, darunter Linux, Tizen, Android, Arduino, Windows, MAC OSX, IOS.

Abbildung 8 zeigt die verschiedenen Geräte-Stacks für die IoTivity-Implementierung. Die Kernfunktionalitäten werden mit dem C SDK implementiert, während zusätzliche Funktionen mit dem C++ SDK realisiert werden. Die Java-Sprachbindung basiert auf dem C++ SDK und imitiert dessen APIs.

Darüber hinaus bietet die neueste IoTivity-Version auch eine AllJoyn Bridge [2] Implementierung, die es ermöglicht, IoTivity mit dem bereits in Windows integrierten AllJoyn-Standard [3] verbinden zu lassen. AllJoyn ist ein ähnlicher Ansatz, der drahtlose Geräte über einen softwarebasierten Bus miteinander verbindet. Auf diese Weise können Geräte aus beiden Familien miteinander kommunizieren.

2.11 Referenzen

[1] Open Connectivity Foundation, "OCF Core Specifiation 1.0.0.0", 2017.

[2] Open Connectivity Foundation, "IoTivity Wiki - Alljoyn Bridge", 18. August 2017. [Online]. Verfügbar: https://wiki.iotivity.org/projects_and_functions#alljoyn-bridge. [Zugriff: 04.12.2017].

Microsoft Corporation, "Windows IoT Core Dokumentation - AllJoyn." [Online]. Verfügbar unter: https://docs.microsoft.com/en-us/windows/iot-core/archive/alljoyn. [Zugriff: 04.12.2017].

[4] B. Konieczek, M. Rethfeldt, F. Golatowski und D. Timmermann, "Ein verteilter Zeitserver für die Echtzeit-Erweiterung von CoAP", im *19. IEEE International Symposium on Real-Time Distributed Computing*, 2016, S. 84-91.

[5] K. Hartke, "Observing Resources in the Constrained Application Protocol (CoAP)", Sep. 2015.

[6] G. Gorbach, "IoT-Standards bekommen einen großen Schub: Treffen Sie die Open Connectivity Foundation (OCF)," 23-Feb-2016. [Online]. Verfügbar unter: https://industrial-iot.com/2016/02/meet-the-open-connectivity-foundation-ocf/. [Zugriff: 04-Aug-2017].

[7] Open Connectivity Foundation, "OCF-Mitgliederliste". [Online]. Verfügbar unter: https://openconnectivity.org/foundation/membership-list. [Zugriff: 04-Aug-2017].

[8] K. Elfström, "Bewertung der IoTivity: Eine Middleware-Architektur für das Internet der Dinge", KTH Royal Institute of Technology, 2017.

[9] Open Connectivity Foundation, "IoTivity Dokumentation - Programmierhandbuch". [Online]. Verfügbar: https://www.iotivity.org/documentation/linux/programmers-guide. [Zugriff: 15-Sep-2017].

[10] Open Connectivity Foundation, "IoTivity Documentation - Certificate Key Manager - Programmer's Guide". [Online]. Verfügbar unter: https://wiki.iotivity.org/ckm. [Zugriff: 15-Sep-2017].

3 LWM2M

3.1 Allgemeine Beschreibung

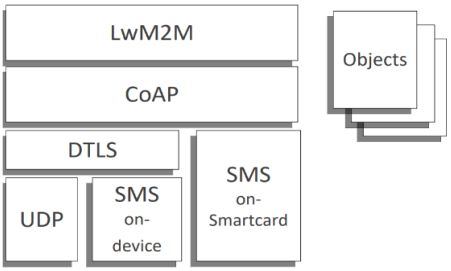

Abbildung 9 Der Protokollstapel des LWM2M Enabler [11].

TPDU (140 bytes)			
DTLS (29 bytes)			CoAP + Effective Payload
Header (13)	Nonce (8)	ICV (8)	
		CoAP (4 bytes)	Effective Payload (107 bytes)

Abbildung 10 LWM2M Request/Response SMS Frame (ohne Token) [11].

LWM2M ist ein Internet of Things (IoT) Gerätemanagement und Machine-to-Machine (M2M) Framework, das von der Open Mobile Alliance (OMA) [1] spezifiziert wurde. Es bietet Geräteverwaltungsfunktionen wie Remote Service Enablement und Remote Application Management zur Wartung, Steuerung und Verwaltung von Remote Embedded Devices und angeschlossenen Geräten im Internet der Dinge (IoT).

Es verwendet das Constrained Application Protocol (CoAP) für den Transport und läuft über DTLS/UDP oder Short Message Service (SMS). Das Protokoll, das die Kommunikation auf der Anwendungsschicht zwischen LWM2M-fähigen Geräten beschreibt, wird als LWM2M Enabler bezeichnet (Abbildung 9). CoAP funktioniert wie HTTP, ist aber speziell für eingeschränkte Geräte konzipiert, z.B. Geräte mit Low-Power-Mikrocontrollern, kleinen Mengen an RAM und Flash und begrenzter Bandbreite. Gleichzeitig kann LWM2M auch mit leistungsfähigeren Embedded-Geräten eingesetzt werden, die von einer effizienten Kommunikation profitieren. Der Header wird minimiert, um den Overhead zu reduzieren, was es möglich macht, Inhalte über einzelne UDP-Datagramme oder Verbindungen mit einer Low Maximum Transmission Unit (MTU) wie SMS in Mobilfunknetzen zu übertragen. SMS-Nachrichten sind auf eine 140 Byte lange Transportprotokoll-Dateneinheit (TPDU) [2] beschränkt. Wie inAbbildung 10 zu sehen ist, nimmt CoAP (4 Bytes Header) zusammen mit DTLS (13 Bytes Header, 8 Bytes nonce, 8 Bytes Integritätsprüfwert (ICV)) 33 Bytes ein und lässt 107 Bytes Nutzlast für LWM2M zur Verwendung übrig. Dies setzt voraus, dass das CoAP-Token nicht verwendet wird, was zusätzlich 8 Bytes aufnehmen und die effektive Payload auf 99 Bytes reduzieren kann. Das CoAP-Token wird verwendet, um Response-Nachrichten mit Requests abzugleichen, wenn mehrere Clients mit einem einzelnen Server kommunizieren. Der Payload besteht aus den verschiedenen LWM2M-Operationen, die auf CoAP-Methoden (GET, PUT, POST, DELETE) abgebildet werden.

Die LWM2M-Netzwerkumgebung besteht aus Clients und Servern mit unterschiedlichen Schnittstellen, wie inAbbildung 12. Die LWM2M-Spezifikationen behandeln die Definition eines Client/Server-Modells anders als üblich. LWM2M Clients laufen auf den Remote-Geräten, die von den LWM2M-Servern aus verwaltet und angesprochen werden.

- LWM2M Clients laufen auf den Endgeräten und ermöglichen den Zugriff auf ein standardisiertes Datenmodell. Einzelne Ressourcen innerhalb des Datenmodells stellen Informationen dar. Sie werden durch einen Endpoint Client Namen identifiziert, der

unabhängig von der Netzwerkadresse eindeutig vergeben wird. Clients werden typischerweise als Softwarebibliothek oder als integrierte Funktion eines Moduls auf dem Gerät integriert.

- LWM2M-Server können die von Clients bereitgestellten Datenmodelle lesen und schreiben. Sie halten eine Verbindung zu den Clients aufrecht und verwalten und überwachen die Geräteressourcen. Die zugreifenden Datenmodelle können so konfiguriert werden, dass nur bestimmte Server Zugriff auf bestimmte Teile der Ressourcen haben. Server werden in der Regel in Rechenzentren gehostet und von M2M-Dienstleistern, Anwendungsdienstleistern oder Netzwerkdienstleistern gewartet.

Für den Datenaustausch definiert LWM2M vier logische Schnittstellen:

- Bootstrap-Schnittstellen initialisieren die Datenmodelle und Verbindungen zu Servern. Ihr einziger Zweck ist es, Clients beim Booten auf die Kommunikation mit normalen LWM2M-Servern vorzubereiten und dafür einen separaten Befehlssatz zu verwenden.
- Registrierungsschnittstellen werden verwendet, um es Clients zu ermöglichen, Server über ihre Anwesenheit und Verfügbarkeit zu informieren.
- Objekt-/Ressourcenzugriffsschnittstellen sind die Hauptschnittstelle, über die die eigentliche Geräteverwaltung erfolgt. Hier können Server verfügbare Objekte/Ressourcen eines Clients ermitteln und darauf zugreifen.
- Berichtsschnittstellen werden für Benachrichtigungen verwendet, bei denen Server bestimmte Ressourcen im Datenmodell des Clients abonnieren und regelmäßig über deren Inhalt informiert werden können.

Das Datenmodell wird durch den LWM2M Enabler definiert und ist ein einfaches Ressourcenmodell, bei dem Informationen als einzelne Ressourcen dargestellt werden. Ressourcen sind Softwarekomponenten wie Datenstrukturen oder diskreter Code. Es kann ein Temperatursensorwert zum Lesen und Überwachen der Wärmeabstrahlung eines Prozessors oder ein Spannungsstellgliedswert zum Ändern und Steuern der Kernspannung sein. Die Ressourcen sind weiter in logischen Objekten organisiert, wie inAbbildung 11. Auf jede Ressource wird über einen eindeutigen Resource Identifier (URI) zugegriffen. Ein erster Satz von Objekten ist bereits in der LWM2M-Standardspezifikation enthalten und sollte standardmäßig auf den Clients konfiguriert werden. Sie enthalten Informationen über das Gerät selbst, seinen Standort im Netzwerk, die LWM2M-Firmware, Sicherheitsrichtlinien, Zugangskontrollrichtlinien und serverbezogene Konfigurationen, die die zu verwaltenden Daten und Funktionen definieren. Ein Beispiel für das Geräteobjekt könnte die Fähigkeit von Servern beinhalten, Informationen wie Hersteller-, Modell-, Fehler- oder Leistungsinformationen abzurufen.

3.2 Ebene

Der LWM2M Server stellt die Schnittstelle zu M2M-Anwendungen zur Verfügung, während der LWM2M-Client die Schnittstelle ist, über die Endgeräte ihre Status- und Steuerungsinformationen bereitstellen. Wie inAbbildung 9 sehen ist, befindet sich der gesamte LWM2M Enabler und seine Endpunkte auf einer High-Layer-Schicht und kann über die vorhandenen Open-Source-Bibliotheken in jede M2M-Anwendung integriert werden.

3.3 Standardisierung

LWM2M ist standardisiert und wird in verschiedenen Open-Source-Projekten eingesetzt. Darüber hinaus ist die OMA ein wichtiges Normungsgremium, das ständig neue offene Standards für die Mobilfunkindustrie entwickelt. Die meisten großen Mobilfunk-Systemhersteller wie Samsung, Sony, Huawei, Mobilfunkbetreiber wie Vodafone, Verizon und Softwareanbieter wie Microsoft sind Teil der Organisation [10]. Der LWM2M-Standard wurde als Reaktion auf den Wunsch der Industrie nach

kostengünstiger Fernverwaltung und Servicefähigkeit über drahtlose Verbindungen entwickelt. Es wird daher erwartet, dass der Großteil der Industrie für eingebettete mobile Geräte in der Automatisierung den LWM2M-Standard unterstützt.

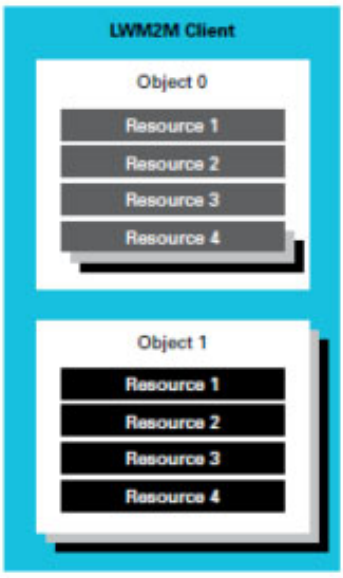

Abbildung 11 Das LWM2M-Datenmodell [12]

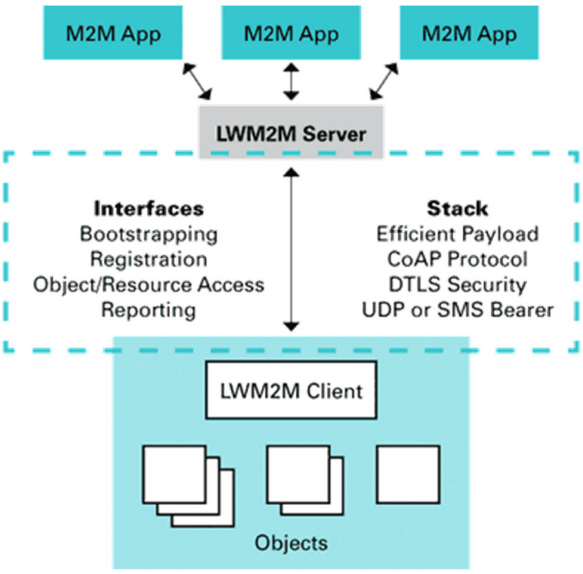

Abbildung 12 Die LWM2M-Architektur [13]

3.4 Architektur

LWM2M stellt seine Daten in Datenmodellen dar. Diese bestehen aus Objekten, die unterschiedliche Ressourcen und Attribute enthalten. Obwohl die Terminologie eine objektorientierte Architektur (OOA) vorschlägt, ist das LWM2M-Framework näher an einer ressourcenorientierten Architektur (ROA). Die Objekte sind nur die oberste Ebene eines hierarchischen Ressourcenbaums. In einer OOA initiieren Anwendungen eine Objektinstanz und greifen auf Methoden oder Datenstrukturen innerhalb dieses Objekts zu. Die Kommunikation ist daher zustandsorientiert, da die Objektinstanz ihre Zustandsinformationen auf dem Server speichern muss (bei LWM2M wäre es der Client). In einer ROA werden Informationen als statische Ressourcen aus externen Services abgerufen, verarbeitet und dem Benutzer präsentiert. Die Kommunikation in einer ROA ist zustandslos, da die Kommunikation zwischen Server und Client immer alle Informationen enthält, die zur Ausführung der Anfrage benötigt werden. Während dies bei der CoAP-Schicht der Fall ist, sind die hinzugefügten LWM2M-Funktionalitäten darüber hinaus nicht völlig zustandslos. Clients müssen von Servern entdeckt werden, bevor die Kommunikation beginnen kann, und die Strukturen der Datenmodelle müssen ausgetauscht werden, damit Server auf die Ressourcen zugreifen können.

3.5 Messaging-Paradigma

LWM2M kapselt alle seine Befehle, Ressourcenanforderungen, Registrierungen und Berichte in einzelnen CoAP-Request/Response-Nachrichten. Die Reporting-Schnittstelle ermöglicht aber auch die Verwendung der CoAP Observer Extension [9]. Es ermöglicht das periodische Abrufen einer Ressource ohne die Komplexität und den großen Aufwand, die durch das regelmäßige HTTP-Polling verursacht werden.

3.6 Echtzeitfähigkeit

LWM2M ist vollständig in CoAP gekapselt. Da CoAP keine sofort einsatzbereiten Echtzeit-(RT)-Funktionen bietet, ist auch LWM2M betroffen. CoAP verwendet das klassische Request/Response-Modell, das für die RT-Kommunikation keinen optimierten Ansatz darstellt, da die Art der Round-Trip-Kommunikation zusätzliche Verzögerungen verursacht. Es gibt Versuche, eine vollständig deterministische CoAP-Kommunikation zu erreichen [8], aber dies sind experimentelle Erweiterungen und sind noch nicht Teil der offiziellen Standardspezifikation.

3.7 Servicequalität

Ähnlich wie bei den Echtzeitaspekten ist LWM2M an die QoS-Funktionen von CoAP gebunden. Regular CoAP hat eingeschränkte QoS-Fähigkeiten. Es verwendet Nachrichten-IDs, um nach Duplikaten zu suchen, und bietet Nachrichtentypen, die bestätigt werden müssen. Die IDs werden verwendet, um Nachrichten und deren Bestätigung zu verfolgen. Dies bietet eine sehr einfache verbindungsorientierte QoS.

3.8 Sicherheit

Wie inAbbildung 9 zu sehen ist, wird DTLS verwendet, um die Kommunikation zwischen LWM2M-Clients und -Servern über UDP zu sichern. Sowohl Pre-Shared als auch Public Key DTLS-Sicherheitsmodi sind verfügbar, um eingeschränkte und leistungsfähigere eingebettete Geräte zu unterstützen. Zugriffskontrolle, Schlüsselverwaltung und alle weiteren Konfigurationen werden vom Bootstrapper übernommen. Schlüssel, Endpunktkennungen und Sicherheitskennungen, die beim Verbindungsaufbau definiert werden, werden in der gesamten LWM2M-Architektur einheitlich verwendet.

3.9 Physikalische Anforderungen

LWM2M basiert auf CoAP, das für den Einsatz auf eingeschränkten Geräten konzipiert ist. Es ist auf einen sehr geringen Overhead optimiert. Darüber hinaus basiert das LWM2M-Datenmodell auf einem sehr einfachen Design, das leistungsstarke Lese-/Schreiboperationen ermöglicht und sehr wenig Rechenleistung benötigt.

3.10 Verfügbare Implementierungen

LWM2M ist ein offener Standard und steht als Open Source Implementierung in verschiedenen Projekten zur Verfügung. Leshan ist eine Java-Client/Server-Implementierung [3], während Anyjay eine C-Client-Implementierung [4] bereitstellt. Weitere Beispiele sind die Awa C Client/Server Implementierung [5] oder die mbed C++ Client/Server Implementierung [6]. Es gibt auch ein Addon für Mozilla Firefox, das den Zugriff auf LWM2M-Server direkt aus dem Browser ermöglicht [7].

3.11 Referenzen

[1] Open Mobile Alliance, "LightweightM2M Spezifikationen", 18. Juli 2017. [Online]. Verfügbar unter: http://www.openmobilealliance.org/release/LightweightM2M/. [Zugriff: 12-Sep-2017].

[2] E. Wilde und A. Vaha-Sipila, "Short Message Service (SMS)", Jan. 2010.

Finsternis, "Finsternis Leshan", 2017. [Online]. Verfügbar unter: http://projects.eclipse.org/projects/iot.leshan. [Zugriff: 12-Sep-2017].

4] AVSystem, "Anjay LwM2M Bibliotheksdokumentation", 2017. [Online]. Verfügbar unter: https://avsystem.github.io/Anjay-doc/. [Zugriff: 12-Sep-2017].

[5] "AwaLWM2M Repository." [Online]. Verfügbar unter: https://github.com/ConnectivityFoundry/AwaLWM2M. [Zugriff: 12-Sep-2017].

[6] Arm Limited, "Mbed Client". [Online]. Verfügbar unter:

https://www.mbed.com/en/platform/mbed-client/. [Zugriff: 12-Sep-2017].

[7] OpenMobileAlliance, "OMA LWM2M DevKit Repository." [Online]. Verfügbar unter: https://github.com/OpenMobileAlliance/OMA-LWM2M-DevKit. [Zugriff: 12-Sep-2017].

[8] B. Konieczek, M. Rethfeldt, F. Golatowski und D. Timmermann, "Ein verteilter Zeitserver für die Echtzeit-Erweiterung von CoAP", im *19. IEEE International Symposium on Real-Time Distributed Computing*, 2016, S. 84-91.

[9] K. Hartke, "Observing Resources in the Constrained Application Protocol (CoAP)", Sep. 2015.

[10] Open Mobile Alliance, "Aktuelle OMA-Mitglieder". [Online]. Verfügbar unter: http://openmobilealliance.org/membership/current-members. [Zugriff: 12-Sep-2017].

[11] Open Mobile Alliance, "Lightweight Machine to Machine Technische Spezifikation - Genehmigte Version 1.0", Feb. 2017.

[12] J. Prado, "OMA Lightweight M2M Resource Model", 2016. [Online]. Verfügbar unter: https://www.iab.org/activities/workshops/iotsi/. [Zugriff: 12-Sep-2017].

[13] P. Desai, "Was ist LWM2M?", 30-Jun-2017. [Online]. Verfügbar unter: https://docs.litmusautomation.com/pages/viewpage.action?pageId=1180863. [Zugriff: 27-Nov-2017].

4 AllJoyn

4.1 Allgemeine Beschreibung

AllJoyn ist ein Open-Source-Framework, das Mittel zur Vernetzung verteilter Anwendungen bereitstellt, die auf getrennten Geräten auf verschiedenen Betriebssystemen oder Hardwareplattformen laufen. Es zielt darauf ab, die Probleme zu lösen, die mit dem Zusammenschluss heterogener verteilter Systeme in einer hochmobilen Umgebung verbunden sind. Während AllJoyn einen hohen Fokus auf intelligente Heimautomation im Consumer-Bereich legt, könnte es bei der Entwicklung heterogener IoT-Systeme in der industriellen Welt helfen. "Selbstbewusste" Netzwerke, die ständig erfahren, welche neuen Geräte hinzugefügt wurden und welche Fähigkeiten oder Schnittstellen zur Verfügung stehen, könnten sich dynamisch anpassen. Das Prinzip von AllJoyn besteht darin, dass physisch getrennte Geräte so kommunizieren können, als wären sie Teil desselben physischen Systems. Dazu gehört auch der Austausch von Nachrichten wie Methodenaufrufe, Antworten oder Signale. Ein Programm, das in einem anderen Adressraum auf einer anderen Maschine läuft, kann so aufgerufen werden, als wäre es lokal. Darüber hinaus können Statusinformationen mit Signalmeldungen ausgetauscht werden. Dies wird durch den AllJoyn-Bus erreicht, eine softwarebasierte Lösung, die die Geräte virtuell miteinander verbindet. Jedes Gerät stellt seine Prozesse einem eigenen lokalen AllJoyn-Bus zur Verfügung, der als Daemon läuft, wie inAbbildung 13 dargestellt. Daemons auf verschiedenen Geräten können sich gegenseitig entdecken und sich über ein Netzwerk logisch verbinden (Abbildung 14). Der Ansatz basiert auf dem Desktop-Bus (D-Bus) Drahtprotokoll [1]. D-Bus wird verwendet, um einen Softwarebus bereitzustellen, der die Interprozesskommunikation (IPC) und Remote Procedure Calls (RPC) zwischen verschiedenen Computerprogrammen, die auf derselben Maschine laufen, ermöglicht. AllJoyn wendet das gleiche Prinzip auf die drahtlose Domäne an. Hochrangige Anwendungen sind sich der physikalischen Trennung zwischen den Geräten nicht bewusst und können Methoden aufrufen und Signale untereinander austauschen, als wären sie Teil desselben Busses.

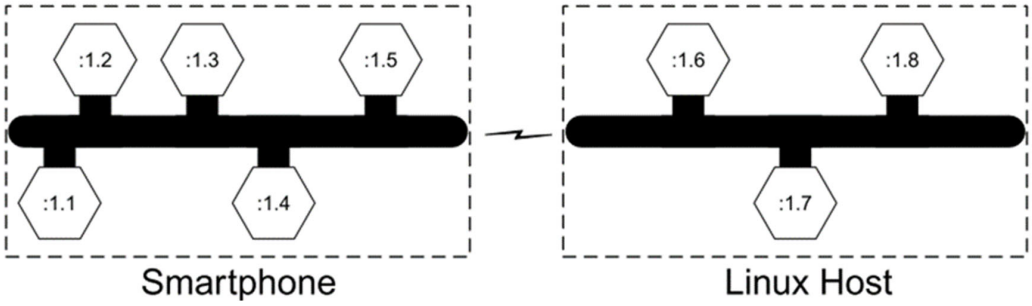

Abbildung 13 AllJoyn-Busse von zwei verschiedenen Vorrichtungen, die verschiedene Prozesse verbinden, die als Hexagone dargestellt sind. Die Prozesse werden über ihre Verbindungsnamen angesprochen und sind in dieser Abbildung vereinfacht (z.B. :1.2) [10].

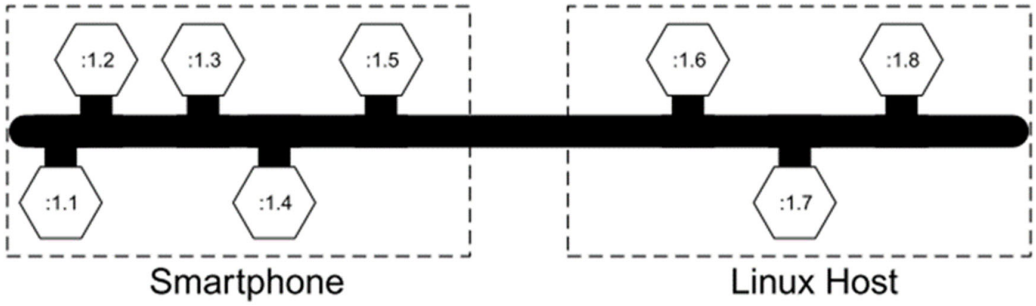

Abbildung 14 Verbundene AllJoyn-Busse von zwei physikalisch getrennten Geräten [10].

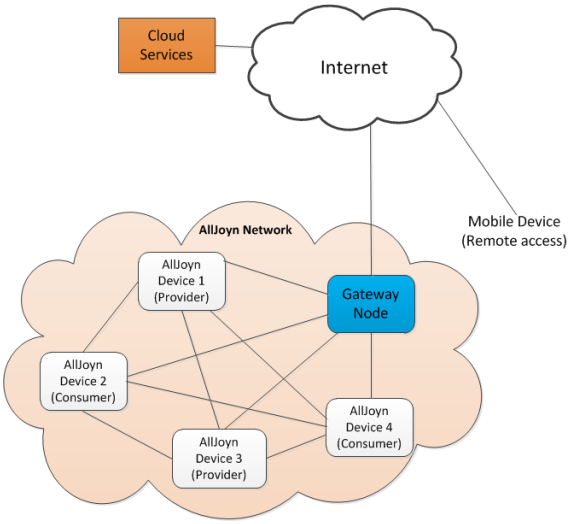

Abbildung 16 Fernzugriff auf die AllJoyn-Netzwerkarchitektur [2].

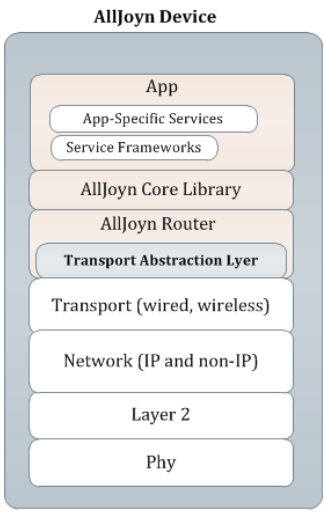

Abbildung 15 AllJoyn Protokollstapel [2].

AllJoyn hat sich zum Ziel gesetzt, das Problem zu lösen, dass mobile Geräte in die Nähe anderer Geräte gelangen und gekoppelt werden müssen, um kommunizieren zu können. Durch die Nutzung eines Advertise- und Discovery-Dienstes identifizieren und verbinden sich die Geräte automatisch über die zugrundeliegenden Netzwerke. Ein Anwendungsfall für eine mobile Umgebung wäre das automatische Zusammenfügen von Geräten, die mit denselben drahtlosen Netzwerken (Wi-Fi oder Bluetooth) verbunden sind. AllJoyn-Netzwerke, die aus kommunizierenden Geräten in der Nähe bestehen,

können auch über einen Gateway-Knoten aus der Ferne angesprochen werden (Abbildung 16). Die von den Geräten bereitgestellten Dienste werden vom Gateway-Knoten über Standard-APIs im Internet-Stil wie REST einem bestehenden Cloud-basierten Dienst bereitgestellt.

4.2 Ebene

AllJoyn arbeitet auf hohem Niveau und ist damit hardwareplattformunabhängig. Es bietet eine Abstraktionsschicht, die sich auf der Anwendungsschicht im OSI-Schichtmodell befindet (Abbildung 15). Die oberste Ebene umfasst eine Anwendung, die Dienste auf der Anwendungsebene bereitstellt und Service-Frameworks unterstützt. Die untenstehende Kernbibliothek ermöglicht es Anwendungen, die wichtigsten AllJoyn-Funktionen aufzurufen. Der AllJoyn Router implementiert die Kernfunktionalitäten für Advertising und Discovery.

4.3 Standardisierung

AllJoyn wurde von Qualcomm entwickelt und wird heute von der AllSeen Alliance vertrieben, einer gemeinnützigen Organisation, die sich der breiten Einführung von Systemen, Dienstleistungen und Produkten zur Unterstützung des IoT verschrieben hat. Seine Mitglieder wie Microsoft und LG bieten AllJoyn-Kompatibilität in ihren Produkten. Die AllSeen Alliance fusioniert derzeit mit dem OCF (Open Connectivity Foundation). Es gehört zu den größten Organisationen für industrielle Konnektivitätsstandards für IoT [6], bestehend aus Mitgliedern wie Samsung, Intel und Cisco [7]. Eine zukünftige Unterstützung in industriellen Anwendungen und eine breite Akzeptanz sind daher sehr wahrscheinlich.

4.4 Architektur

Das AllJoyn-Framework ist ein objektorientiertes System. Aus der Perspektive der Framework-Schicht ermöglicht AllJoyn den IPC, bei dem kommunizierende Geräte eng miteinander verbunden sind und der Informationsaustausch zustandsorientiert ist. Während Transportschichtprotokolle wie UDP von Natur aus zustandslos sind, müssen AllJoyn-fähige Geräte die Schnittstellen des anderen kennen, bevor die Kommunikation eingeleitet werden kann, und sind daher auf der Anwendungsschicht zustandsorientiert.

4.5 Messaging-Paradigma

Während der größte Teil der Kommunikation auf Methodenaufrufen und dem Signalaustausch über IPC und RPC basiert, stellt AllJoyn auch eine Observer-API zur Verfügung, die auf publish/subscribe Weise arbeitet, aber anstatt den Anwendungen zu helfen, eigene Funktionalität bereitzustellen, erleichtert sie vielmehr den Verbrauch von Diensten oder Informationen, die von den Peers auf dem AllJoyn-Bus angeboten werden. Die Grundidee besteht darin, eine Reihe von erforderlichen Schnittstellen zu abonnieren und den Client zu benachrichtigen, wenn ein Peer auf dem Bus erscheint oder verschwindet, wenn diese Schnittstellen bereitgestellt werden [5]. Ziel ist es, Objekte auf einem Bus zu entdecken, die eine Reihe von gegebenen Schnittstellen implementieren und anschließend mit ihnen interagieren.

4.6 Echtzeitfähigkeit

Wie die QoS-Aspekte sind auch die Echtzeitfähigkeiten abhängig von den verwendeten Transportprotokollen innerhalb der kommunizierenden Anwendung. AllJoyn arbeitet mit TCP- oder UDP-Transport.

4.7 Servicequalität

Da AllJoyn im OSI-Schichtmodell oberhalb der Transportschicht arbeitet, hat es keinen Einfluss auf das verwendete zugrundeliegende Transportprotokoll. Die QoS-Optionen hängen vom verwendeten zugrundeliegenden Transportprotokoll ab. Je nachdem, ob die Anwendung verbindungslose oder

verbindungsorientierte Dienste benötigt, kann UDP oder TCP verwendet werden. Wenn das Hauptanliegen einer verteilten Anwendung die Fähigkeit ist, große Datenmengen zuverlässig zu übertragen, sollte der TCP-Transport in Betracht gezogen werden. Besteht das Verkehrsmuster hauptsächlich aus Bursts signifikanter Datenmengen zwischen längeren Leerlaufzeiten, sollte stattdessen der UDP-Transport berücksichtigt werden [4].

4.8 Sicherheit

AllJoyn bietet ein Sicherheitsframework. Wie inAbbildung 18 zu sehen ist, werden Authentifizierung und Datenverschlüsselung auf der Anwendungsschicht realisiert. Jede kommunizierende Anwendung wird durch ihre Schnittstelle im AllJoyn-Bus identifiziert. Für eine Authentifizierung wird jeder Schnittstelle ein Globally Unique Identifier (GUID) zugeordnet, der für die Speicherung und den Zugriff auf Authentifizierungs- und Verschlüsselungsschlüssel für die zugehörige Anwendung verwendet wird. Für die Authentifizierung wird das Sicherheits-Framework Simple Authentication and Security Layer (SASL) verwendet. In den Schlüsselspeichern befinden sich die GUID's und alle benötigten Geheimschlüssel für die Datenverschlüsselung. Es wird vom Sicherheitsmodul verwaltet, das in der AllJoyn Core Library implementiert ist, und verwendet die *Auth Listener* Callback-Funktion in der Anwendung, um Authentifizierungsdaten zu erhalten und Zertifikate zu überprüfen. Die AllJoyn-Router tauschen nur Sicherheitsmeldungen aus, die Managementinformationen zum Aufbau und zur Aufrechterhaltung einer sicheren Verbindung enthalten (z.B. Zertifikate, Schlüssel etc.), implementieren aber selbst keine Sicherheitslogik.

Security 2.0 ist eine Erweiterung der bestehenden AllJoyn Security. Darüber hinaus ermöglicht es dem Eigentümer, Richtlinien zu installieren, die den Zugriff auf sichere Schnittstellen oder Objekte individuell validieren können [8].

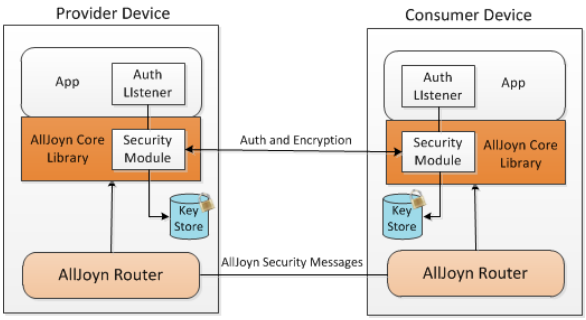

Abbildung 18 AllJoyn Sicherheitsarchitektur [11]

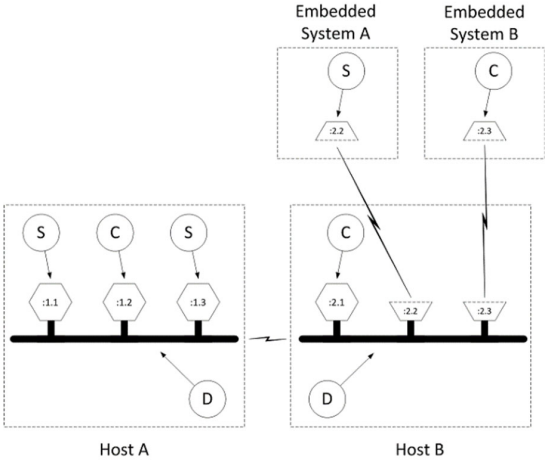

Abbildung 17 AllJoyn Distributed Bus mit Thin-Core-Bibliotheken [9].

4.9 Physikalische Anforderungen

Die AllJoyn Standard Core Library (AJSCL) wurde entwickelt, um in einer objektorientierten Umgebung auf Universalcomputern mit Windows, Linux usw. ausgeführt zu werden. Um AllJoyn auf eingeschränkten eingebetteten Geräten zu ermöglichen, wurde die AllJoyn Thin Core Library (AJTCL) [9] eingeführt. Es reduziert alle AJSCL-APIs in eine kompaktere und C-basierte Form oder führt sie remote auf einem anderen leistungsfähigeren System aus. So erfordern z.B. die Routing-Funktionalitäten Multithreading und große Mengen an RAM und ROM und bieten somit die Möglichkeit, remote zu arbeiten. Ein Beispiel ist in Abbildung 17 zu sehen, wo Dienste (S) und Clients (C) auf den eingebetteten Systemen A und B Router von Host B leihen, die eine andere Maschine repräsentieren. In diesem Beispiel könnte Host B eine Brücke sein, die mit einem Smartphone Host A

über die angeschlossenen Bussegmente (D) kommuniziert. Die Brücke könnte die Routing-Funktionalitäten und den Buszugang für intelligente Leuchten oder Temperatursensoren beinhalten. Ein Client auf Host A könnte den aktuellen Temperaturwert vom Sensordienst auf dem Embedded System A anfordern. Oder der Client auf dem Embedded System B könnte eine intelligente Leuchte sein, die aktiviert wird, sobald Host A außer Reichweite ist und sich vom Bus trennt.

4.10 Verfügbare Implementierungen

Das AllJoyn SDK ist ein Open-Source-Produkt und wird von der AllSeen Alliance für Windows, Linux, Android, iOS, OS X und OpenWRT [3] bereitgestellt. Anwendungen, die in C, C++, C#, Objective C und Java geschrieben wurden, haben Sprachbindungen in AllJoyn unterstützt.

4.11 Referenzen

[1] H. Pennington, A. Carlsson, A. Larsson, S. Herzberg, S. McVittie und D. Zeuthen, "D-Bus Spezifikation". [Online]. Verfügbar unter: https://dbus.freedesktop.org/doc/dbus-specification.html. [Zugriff: 15-Aug-2017].

[2] AllSeen Alliance, "Dokumentation - Systemarchitektur". [Online]. Verfügbar unter: https://allseenalliance.org/framework/documentation/learn/core/system-description/system-architecture. [Zugriff: 04-Aug-2017].

[3] AllSeen Alliance, "AllJoyn SDK Download." [Online]. Verfügbar unter: https://allseenalliance.org/framework/download. Zugriff: 14-Aug-2017].

AllSeen Alliance, "ALLJOYN® TRANSPORTET BESTE PRÄKTE", 2017. [Online]. Verfügbar unter: https://identity.allseenalliance.org/developers/develop/api-guide/core/alljoyn-transports-best-practices. [Zugriff: 04.12.2017].

[5] AllSeen Alliance, "Dokumentation - Beobachter". [Online]. Verfügbar unter: https://allseenalliance.org/framework/documentation/develop/api-guide/core/observer. Zugriff: 14-Aug-2017].

[6] G. Gorbach, "IoT-Standards bekommen einen großen Schub: Treffen Sie die Open Connectivity Foundation (OCF)," 23-Feb-2016. [Online]. Verfügbar unter: https://industrial-iot.com/2016/02/meet-the-open-connectivity-foundation-ocf/. [Zugriff: 04-Aug-2017].

[7] Open Connectivity Foundation, "OCF-Mitgliederliste". [Online]. Verfügbar unter: https://openconnectivity.org/foundation/membership-list. [Zugriff: 04-Aug-2017].

[8] AllSeen Alliance, "Dokumentation - Sicherheit 2.0." [Online]. Verfügbar unter: https://allseenalliance.org/framework/documentation/learn/core/security2_0/hld. Zugriff: 14-Aug-2017].

[9] AllSeen Alliance, "Dokumentation - AllJoyn Thin Core." [Online]. Verfügbar unter: https://allseenalliance.org/framework/documentation/learn/core/thin-core. Zugriff: 14-Aug-2017].

AllSeen Alliance, "Einführung in das AllJoyn ™ Framework", Dez. 2013.

[11] AllSeen Alliance, "Dokumentation - Sicherheit". [Online]. Verfügbar unter: https://allseenalliance.org/framework/documentation/learn/core/system-description/alljoyn-security. Zugriff: 14-Aug-2017].

5 Open Platform Communication Unified Architecture (OPC UA)

5.1 Allgemeine Beschreibung

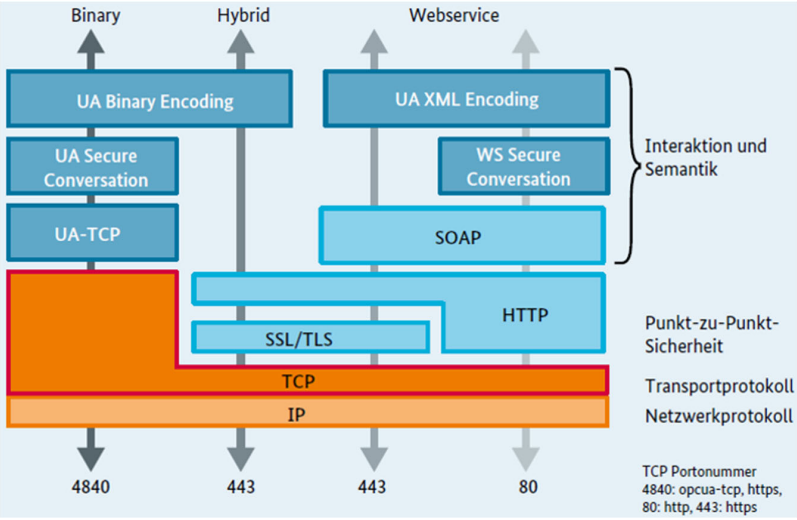

Abbildung 19 Optionen des OPC UA-Protokolls [2].

Die Open Platform Communication Unified Architecture (OPC UA) ist eine plattformunabhängige, serviceorientierte Architektur für verteilte Automatisierungssysteme mit sicheren Transportmechanismen und Datenmodellierung. Sie kann als Schnittstelle zwischen Automatisierungssystemen in verschiedenen Ebenen der Automatisierungspyramide [1] eingesetzt werden. DieAbbildung 19 Optionen des zeigt verschiedene OPC UA Protokolloptionen, die die Kompatibilität mit verschiedenen Kommunikationsprotokollen in industriellen Netzwerken zeigen. Durch die Sicherheitsfunktionen dieser Middleware kann eine durchgängige Sicherheit für die unternehmensübergreifende Kommunikation ermöglicht werden. Die von OPC UA bereitgestellten Informationsmodellierungsstandards können die Machbarkeit der Implementierung eines Standarddatenmodells für den Informationsaustausch erhöhen. Es gibt mehrere Anwendungsfälle für diese Architektur, z.B. verwendet die Überwachung eines Offshore-Windparks OPC UA Server zur Datenbereitstellung für SCADA-Systeme (Global Tech I-Projekt) [4].

5.2 Ebene

OPC UA ist ein Middleware-Framework, das verschiedene Protokolloptionen von der Transportschicht bis zu höheren Schichten unterstützt (vgl.Abbildung 19 Optionen des).

5.3 Standardisierung

OPC UA ist in der IEC 62541 mit 13 Spezifikationsteilen standardisiert, von denen die ersten 7 Teile Kernspezifikationen wie Konzepte, Sicherheitsmodell, Adressraummodell, Dienste, Informationsmodell, Mappings und Profile sind. Die anderen Teile sind Zugriffsartenspezifikationen wie Datenzugriff, Alarm und Bedingungen, Programme, Zugriff auf Historie und Aggregate, ein Teil über die Discovery Services. Dies sind verschiedene Teile der Spezifikationen, die derzeit als stabile Version [3] verfügbar sind.

5.4 Architektur

OPC UA ist eine serviceorientierte Architektur. Die Informationsmodellierung ist einer der Vorteile von OPC UA [5]. Im Gegensatz zu klassischem OPC ermöglicht die Informationsmodellierung von OPC UA die Darstellung der Semantik von Daten.

Im Allgemeinen ist es nicht erforderlich, dass ein OPC UA-Client über ein integriertes OPC UA-Informationsmodell verfügt und diese Informationen an einen OPC UA-Server weitergeben muss (siehe Kapitel 2 in[1]).

5.5 Messaging-Paradigma

OPC UA basiert auf einem Request/Response-Paradigma für Client/Server-Anwendungen.

Neben demregulären Nachrichtentyp Request/Response, wurde OPC UA kürzlich um eine Spezifikationen für publish/subscribe erweitert [3].

5.6 Echtzeitfähigkeit

OPC UA TSN (Time Sensitive Networking) ist eine in Arbeit befindliche standardisierte Spezifikation, die die harte Echtzeitkommunikation mit dem Publish/Subscribe-Modell [6], [7], [8] unterstützt. Das Pub/Sub-Modell wird mit dem Real-Time Physical Layer (Ethernet mit TSN) kombiniert, um echtzeitfähige OPC UA [7] zu erreichen.

5.7 Servicequalität

Die Parameter für die Servicequalität basieren auf den Transportprotokollen. Für das allgemeine Servicekonzept bietet OPC UA Timeout-Handling und Fehlerbehandlung (siehe Abschnitt 5.2.1 und 5.2.3 in[1]).

5.8 Sicherheit

Die OPC UA Sicherheitsarchitektur [11] definiert einen mehrschichtigen Ansatz für die Sicherheit, wie in Abbildung 20. dargestellt. Die Anwendungsschicht oben in Abbildung 20 dient der Übertragung von Anlageninformationen, Einstellungen, Anweisungen und Echtzeitdaten von Geräten zwischen einem Client und einem Server während einer Sitzung. Die Sitzung dient der Authentifizierung und Autorisierung der Benutzer, die mit dem Kunden und bestimmten Produkten arbeiten. Eine Sitzung läuft über den SecureChannel. Der SecureChannel verfügt über drei verschiedene Betriebsarten, wie nachfolgend erläutert. Die Integrität wird durch digitale Signaturen und die Vertraulichkeit durch Verschlüsselung der Informationen der übertragenen Nachrichten garantiert. Die unterste Schicht ist die Transportschicht, die die Übertragung von gesicherten Daten über Socketverbindungen ermöglicht (siehe Abschnitt 7.5.1.1.2 [1]).

Nach der Socket-Erstellung gibt es drei Modi, die ein Client beim Öffnen eines sicheren Kanals nutzen kann (siehe Abschnitt 7.5.2 in [1]):

- „None“ - OpenSecureChannel-Nachricht wird nicht gesichert.
- „Sign“ - Die Nachricht wird mit dem zugehörigen privaten Schlüssel des Application Instance Certificate des OPC UA Clients signiert.
- „Sign and Encrypt“- Die Nachricht wird zusätzlich mit dem öffentlichen Schlüssel des Application Instance Certificate des Servers verschlüsselt (d.h. zusätzlich zum Sign-Modus).

Die Zertifikate der Anwendungsinstanz werden in anderen Modi als dem Modus "None" ausgetauscht. Asymmetrische Schlüssel werden verwendet, um die SecureChannel-Nachrichten zu sichern, und symmetrische Schlüssel werden aus Geheimnissen generiert, die zwischen Client und Server während der Einrichtung des SecureChannels geteilt werden. Weitere Nachrichten werden gemäß

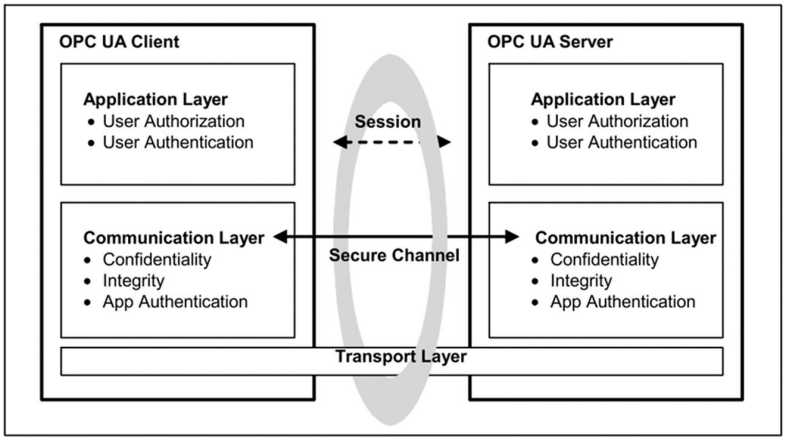

Abbildung 20 OPC UA Sicherheitsarchitektur [11]

MessageSecurityMode und SecurityPolicy gesichert, die mit den erzeugten symmetrischen Schlüsseln ausgehandelt werden.

Die folgenden Arten der Benutzerauthentifizierung können mit dem OPC UA-Protokoll verwendet werden (siehe Abschnitt 7.35 in [12]):

- AnonymousIdentityToken - Es sind keine Benutzerinformationen verfügbar (wenn keine Benutzerauthentifizierung erforderlich ist).
- UserNameIdentityToken - Ein Benutzer, der durch Benutzername und Passwort identifiziert wird.
- X509IdentityToken - Ein Benutzer, der durch ein X509v3-Zertifikat identifiziert wird.
- IssuedIdentityToken - Ein Benutzer, der durch einen WS-SecurityToken identifiziert wird.

Daher sind sowohl die Anwendungsschicht als auch die Kommunikationsschicht von der Standardinfrastruktur zur Verwaltung der in der Kommunikation verwendeten Zertifikate und Schlüsselpaare abhängig. Der OPC UA Standard spezifiziert nicht, wie eine solche Infrastruktur aussieht, da es viele verschiedene Konzepte gibt, die alle von den konkreten Umgebungen und Anforderungen abhängen (siehe Kapitel 7 in [1]). Ein solches Beispiel ist eine PKI (Public Key Infrastructure)[13], die für den Bereich der industriellen Automatisierung verwendet werden kann.

So kann beispielsweise das mehrschichtige Sicherheitsmodell mit PKI [13] implementiert werden, wie inAbbildung 21 dargestellt. Es kann verwendet werden, um den gesicherten Modus der Datenübertragung im OPC UA-Protokoll zu demonstrieren.

Abbildung 21 stellt sich wie folgt dar: Es gibt 3 verschiedene Einheiten in einer PKI, die Certification Authority (CA), die Registration Authority (RA) und die Validation Authority (VA). Die CA stellt das Vertrauensverhältnis zwischen den beiden an einer Kommunikation beteiligten Partnern her, indem sie das signierte Zertifikat für den anfragenden Partner ausstellt. Die RA dient als Registrierungsbehörde, die die Identität dem angeforderten Benutzer/Client zuordnet. Die VA ist für die Überprüfung der Validierung des Zertifikats verantwortlich, das für die vertrauenswürdige Kommunikation verwendet wird. Diese Validierung kann offline oder online erfolgen, z.B. durch Verwendung einer Certificate Revocation List (CRL) [13] oder eines Online Certificate Status Protocol (OCSP) [14]. Die CA an RA und VA delegiert die oben beschriebenen Aufgaben. Die Abbildung 21zeigt auch, dass ein Benutzer die RA um ein signiertes Zertifikat von einer CA bittet, indem er eine Certificate Signing Request (CSR) [15] sendet, dann überprüft die RA die Identität des Benutzers an die CA, woraufhin das CA signierte Zertifikat an den angeforderten Benutzer ausgestellt wird. Nun verwendet der Benutzer das empfangene Zertifikat, um eine vertrauenswürdige Kommunikation mit der

Anwendung zu ermöglichen, die auch der gleichen CA vertraut, die das Benutzerzertifikat ausgestellt hat. Ähnlich wie bei dem vorstehend erläuterten Beispiel kann eine PKI-Infrastruktur für ein Zertifikat aufgebaut werden, das für den Aufbau einer sicheren Kommunikation über das OPC-UA-Protokoll erforderlich ist.

Die Attribute der Zugriffsebene und der Benutzerzugriffsebene sind in [16] beschrieben. Diese Attribute bestimmen die für die Knoten konfigurierten Zugriffsebenen und die durch die Konfiguration der Benutzerzugriffsebenen bestimmte Benutzerberechtigung.

Die in [17] beschriebene Rollenspezifikation ist eines der neuesten Updates der OPC Foundation, um ihre Unterstützung für die rollenbasierte Autorisierung hervorzuheben. Das OPC UA Adressraummodell beschreibt eine Base NodeClass, von der alle anderen NodeClasses abgeleitet sind. Beim Vergleich der möglichen Attribute (d.h. der obligatorischen und optionalen Attribute) einer in Tabelle 2 in [16] definierten Base NodeClass mit Tabelle 7 in [17] werden zusätzliche optionale Attribute zur Unterstützung von Rollen und Zugriffsbeschränkungen betrachtet. Diese Attribute sind nun optional für alle abgeleiteten Nodeklassen. Ihre Vorteile und Herausforderungen liegen in der Klarheit der Umsetzung.

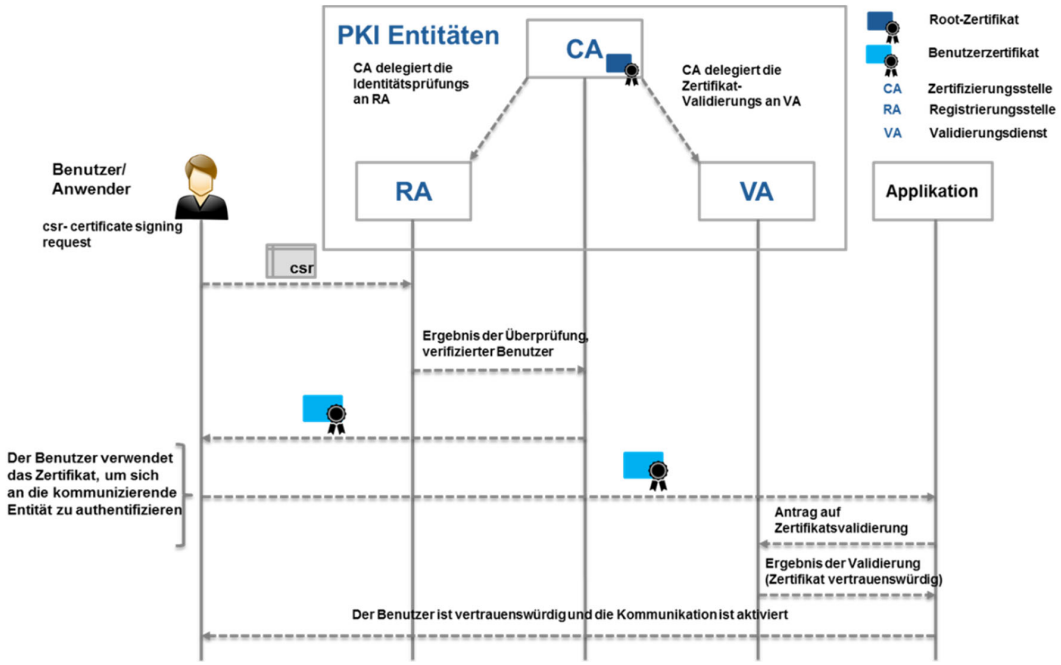

Abbildung 21 Einfache Public-Key-Infrastruktur

5.9 Physikalische Anforderungen

OPC UA unterstützt eingeschränkte Geräte. Die Protokolloptionen bieten binäre Verschlüsselung über TCP, um dies zu erleichtern.

5.10 Verfügbare Implementierungen

Es gibt verschiedene Open Source OPC UA (Stack und SDK) Implementierungen wie Milo (Java) [18], open62541 (C-basierte Bibliothek) [19], Free OPC UA (Python)[20], etc., sowie kommerzielle Produkte wie Prosys OPC (Java) [21], Softing Stack (.NET) [22], etc.). Alle oben genannten Implementierungen haben verschiedene Sätze von OPC UA-Funktionen implementiert, wie in ihren Referenzen angegeben.

Literaturhinweise

Es gibt mehrere Publikationen und Diskussionspapiere, die Anwendungsfälle, Sicherheitsanalysen usw. über OPC UA beschreiben. Eines der Diskussionspapiere von Plattform Industrie 4.0 wie "Sichere Kommunikation für Industrie 4.0 [2]" verwendet den OPC UA als Use Case. Die Veröffentlichung des BSI (Bundesamt für Sicherheit in der Informationstechnik), "OPC UA Security Analysis [23]], beschreibt die Analyse der OPC UA Spezifikationen mit den Teilen 2, 4, 6, 7 und 12. Das Whitepaper mit dem Titel "The OPC UA Security Model for Administrators [24]] diskutiert das Sicherheitsmodell des Frameworks.

5.11 Referenzen

[1] W. Mahnke, S.-H. Leitner und M. Damm, *OPC Unified Architecture*.

[2] Plattform Industrie 4.0, “Sichere Kommunikation für Industrie 4.0,” Bundesministerium für Wirtschaft und Energie (BMWi), 2017.

[3] "OPC Foundation." [Online]. Verfügbar unter: https://opcfoundation.org/about/opc-technologies/opc-ua/. [Zugriff: 17-Aug-2017].

"Volllastprüfstand sorgt für eine reibungslose Inbetriebnahme von Offshore-Windenergieanlagen." [Online]. Verfügbar unter: https://www.pc-control.net/pdf/special_wind_2012/solutions/pcc_special_wind_2012_areva_e.pdf. [Zugriff: 30-Jan-2018].

[5] DIN IEC 62541-5, "OPC Unified Architecture - Teil 5: Informationsmodell". 2015.

[6] "OPC Foundation kündigt Unterstützung von Publish / Subscribe for OPC UA an." [Online]. Verfügbar: https://opcfoundation.org/news/opc-foundation-news/opc-foundation-announces-support-of-publish-subscribe-for-opc-ua/. [Zugriff: 21-Sep-2017].

[7] "Echtzeitfähige OPC UA." [Online]. Verfügbar: https://www.br-automation.com/en/about-us/newsletter/latest-news/real-time-capable-opc-ua/. [Zugriff: 21-Sep-2017].

[8] "Teil 14: PubSub." [Online]. Verfügbar unter: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-14-pubsub. [Zugriff: 21-Sep-2017].

[9] Plattform Industrie 4.0, "Struktur der Administrationsoberfläche", 2015.

Bitkom, VDMA und ZVEI, "Umsetzungsstrategie Industrie 4.0 Bericht über die Ergebnisse der Industrie 4.0 Plattform" S. 1-104, 2016.

[11] DIN IEC 62541-2, "OPC Unified Architecture - Part 2 Security Model". 2015.

[12] DIN IEC 62541-4, "OPC Unified Architecture - Teil 4: Dienstleistungen". 2015.

RFC 5280, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", 2008.

14] RFC 6960, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", 2013.

RFC 2986, "PKCS #10: Certification Request Syntax Spezifikation Version 1.7," 2000.

[16] DIN IEC 62541-3, "OPC Unified Architecture - Teil 3: Adressraummodell", Nov. 2015.

[17] OPC Foundation, "OPC Unified Architecture Specification-Part 3: Address Space Model", Nov. 2017

[18] K. Herron, "Eclipse Milo". [Online]. Verfügbar: https://projects.eclipse.org/projects/iot.milo. [Zugriff: 16-Aug-2017].

[19] OPC Unified Architecture, "open62541." [Online]. Verfügbar unter:

https://github.com/open62541/open62541. [Zugriff: 16-Aug-2017].

[20] OPC Unified Architecture, "LGPL Pure Python OPC-UA Client und Server." [Online]. Verfügbar unter: https://github.com/FreeOpcUa/python-opcua. [Zugriff: 16-Aug-2017].

[21] OPC Unified Architecture, "Prosys OPC." [Online]. Verfügbar unter: https://www.prosysopc.com/. [Zugriff: 16-Aug-2017].

[22] "Softing AG." [Online]. Verfügbar unter: https://industrial.softing.com/en/news/news-details/article//softing-industrial-offers-commercial-license-for-opc-ua-net-standard-stack.html. [Zugriff: 06-Nov-2017].

[23] Bundesamt für Sicherheit in der Informationstechnik (BSI), “OPC UA Security Analysis,” p. 90, 2017.

[24] R. Armstrong und P. Hunkar, "The OPC UA Security Model For Administrators", S. 32, 2010.

6 Streaming Text Oriented Messaging Protokoll (STOMP)

6.1 Allgemeine Beschreibung

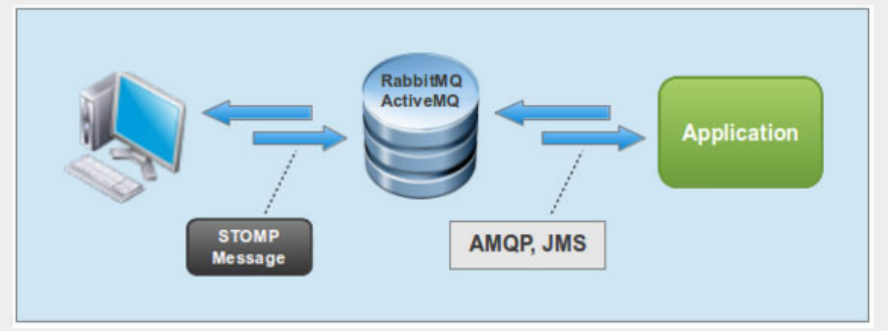

Abbildung 22 Message-Broker-Lösung[3].

STOMP ist ein einfaches interoperables Protokoll, das für den asynchronen Nachrichtenaustausch zwischen Clients über Vermittlungsserver [1] entwickelt wurde. Es wird verwendet, um eine Verbindung zu Nachrichtenbrokern in Unternehmen herzustellen, die Skriptsprachen wie Ruby, Python und Perl verwenden. Das Protokoll ist plattformunabhängig.

Jeder STOMP-Datenrahmen besteht aus einem Befehl, einem Satz optionaler Header und einem optionalen Body. Ein STOMP-Server ist als eine Reihe von Zielen modelliert, an die Nachrichten gesendet werden können, und ein STOMP-Client ist ein Benutzeragent, der in zwei Modi agiert, entweder als Produzent oder als Verbraucher (d.h. entweder Nachrichten an das Ziel senden oder die Nachrichten abonnieren und empfangen). Ein Beispiel für eine Message Broker-Lösung ist inAbbildung 22. dargestellt. Hier ist ActiveMQ (Open Source Messaging and Integration Pattern Server) [2] eine Reihe von Zielen (z.B. JMS-Queues oder Themen, die als eine einzige logische Destination für einen Client zugänglich sind).

6.2 Ebene

STOMP ist ein einfaches, TCP-basiertes Protokoll auf Anwendungsschicht.

6.3 Standardisierung

Die neueste Spezifikationsversion ist STOMP 1.2, veröffentlicht am 22.10.2012 [1].

6.4 Architektur

STOMP ist eine nachrichtenorientierte Middleware.

6.5 Messaging-Paradigma

```
COMMAND
header1:value1
header2:value2

Body^@
```

Abbildung 23 STOMP Datenrahmenstruktur [1]

STOMP verwendet ein Publish/Subscribe-Muster. Es gibt eine begrenzte Liste von Frames sowohl für die Server- als auch für die Client-Seite. Dadurch wird das Verständnis und die Implementierung des Protokolls erleichtert. Alle Befehle und Headernamen sind case-sensitiv. So wird beispielsweise der Rahmen'SEND' nur in dem in der folgenden Tabelle angegebenen Format (Groß-/Kleinschreibung beachten) erkannt.

```
SEND
destination:/queue/a
receipt:message-12345

hello queue a^@
```

Abbildung 24 SEND-Rahmen [1]

Abbildung 23 zeigt einen STOMP-Datenrahmen, der Rahmen beginnt mit einer Befehlszeichenkette, gefolgt von null oder mehr Kopfeinträgen im Format < key> :<value>. Dem Körper folgt dann ein NULL-Oktett. Die Spezifikation[1] verwendet ^@, um das NULL-Oktett darzustellen. So kann beispielsweise ein Client wie in Abbildung 24einen SEND-Frame senden und der Server bestätigt die Verarbeitung des Client-Frames mit einem RECEIPT-Frame.

Die Client-Frames sind folgendermaßen [1] dargestellt,

Rahmen	***Beschreibung***
SENDEN	Der SEND-Rahmen sendet eine Nachricht an ein Ziel im Messaging-System.
ABONNIEREN	Der Rahmen SUBSCRIBE wird verwendet, um sich zu registrieren, um ein bestimmtes Ziel anzuhören.
UNTERBRECHUNG	Der Rahmen UNSUBSCRIBE wird verwendet, um ein bestehendes Abonnement zu entfernen.
ANFANG	BEGIN wird verwendet, um eine Transaktion zu starten.
COMMIT	COMMIT wird verwendet, um eine laufende Transaktion zu bestätigen.
ABBRUCH	ABORT wird verwendet, um eine laufende Transaktion zurückzusetzen.
ACK	ACK wird verwendet, um den Verbrauch einer Nachricht aus einem Abonnement durch eine mandanten- oder mandantenindividuelle Bestätigung zu bestätigen.
NACK	NACK ist das Gegenteil von ACK.
DISCONNECT	Ein Client kann die Verbindung zum Server durch Schließen des Sockets jederzeit trennen, aber es gibt keine Garantie, dass die zuvor gesendeten Frames vom Server empfangen wurden.

Hinweis: Wenn es andere Frames gibt, die nicht in der obigen Liste enthalten sind, die vom Client gesendet werden, kann der STOMP 1.2 [1] Server mit ERROR-Frame antworten und die Verbindung schließen.

Die in der Spezifikation angegebenen Größenbeschränkungen sind wie folgt[1]:

Es gibt Höchstgrenzen für

- Die Anzahl der erlaubten Frame-Header in einem einzigen Frame
- Die maximale Länge der Kopfzeilen
- Die maximale Größe der Rahmenkarosserie

Wenn die Grenzen überschritten werden, sollte der Server dem Client einen ERROR-Frame senden und dann die Verbindung schließen.

Die Server-Frames sind unten [1] dargestellt:

Rahmen	***Beschreibung***
BOTSCHAFT	MESSAGE-Frames werden verwendet, um Nachrichten von Abonnements an den Client zu übermitteln.
EMPFANG	Ein RECEIPT-Frame wird vom Server an den Client gesendet, sobald ein Server erfolgreich einen Client-Frame verarbeitet hat, der einen Empfang anfordert.
FEHLER	Der Server KANN ERROR-Frames senden, wenn etwas schief geht.

6.6 Echtzeitfähigkeit

STOMP unterstützt keine harte Echtzeitkommunikation, sondern eine weiche Echtzeitkommunikation, wenn sie über das WebSockets-Protokoll ausgeführt wird. WebSocket [3] ist ein TCP-basiertes Protokoll, das eine synchrone bidirektionale Kommunikation zwischen einem Client und einem Server ermöglicht. Anstatt einen Strom von Bytes zu übertragen, werden Frames variabler Länge bidirektional ausgetauscht, ohne die Kommunikation nach der Übertragung wie bei TCP zu beenden. Bidirektional bedeutet, dass Daten zwischen den kommunizierenden Parteien jederzeit und ohne vorherige Nachrichtenanforderung gesendet werden können.

6.7 Servicequalität

```
CONNECT
heart-beat:<cx>,<cy>

CONNECTED:
heart-beat:<sx>,<sy>
```

Abbildung 25 Heart Beat Header[1].

Das STOMP-Protokoll hat einen zusätzlichen Header für den Client (vgl. STOMP 1.2 Versionsspezifikation [1]) der als "Heart-beat" bezeichnet wird. Es ist ein optionaler Header. Konzeptionell sind alle neuen Daten, die über die Netzwerkverbindung empfangen werden, ein Hinweis darauf, dass das entfernte Ende am Leben ist, was bedeutet, dass es einen Herzschlag gibt. Zum Beispiel, wenn Herzschläge alle <n> Millisekunden erwartet werden [1]:

- Der Absender muss mindestens alle <n> Millisekunden neue Daten über die Netzwerkverbindung senden.
- Wenn der Absender keinen echten STOMP-Frame zu senden hat, muss er ein Zeilenende (EOL) senden.
- Wenn der Empfänger innerhalb eines Zeitfensters von mindestens <n> Millisekunden keine neuen Daten empfangen hat, darf er die Verbindung als tot betrachten.

- Aufgrund der zeitlichen Ungenauigkeiten sollte der Empfänger tolerant sein und eine Fehlermarge berücksichtigen.

"Heart-beating kann optional verwendet werden, um die Gesundheit der zugrunde liegenden TCP-Verbindung zu testen und sicherzustellen, dass das entfernte Ende am Leben und betriebsfähig ist" [1]. Dies geschieht gleich zu Beginn der STOMP-Sitzung, indem Sie den CONNECT- und CONNECTED-Frames [1] einen Heart-Beat-Header hinzufügen. Der Heart Beat-Header muss zwei positive ganze Zahlen enthalten, die durch ein Komma getrennt sind, wie inAbbildung 25.

Die erste Zahl steht für das, was der Sender des Frames tun kann (ausgehende Herzschläge):

- 0 bedeutet, dass es keine Herzschläge senden kann.
- sonst ist es die kleinste Anzahl von Millisekunden zwischen den Herzschlägen, die es garantieren kann.

Die zweite Zahl stellt dar, was der Sender des Frames erhalten möchte (eingehende Herzschläge):

- 0 bedeutet, dass es keine Herzschläge empfangen möchte.
- ansonsten ist es die gewünschte Anzahl von Millisekunden zwischen den Herzschlägen.

6.8 Sicherheit

Die Sicherheitsaspekte sind nicht Bestandteil der STOMP-Spezifikation [4]. Sicherheitskonfigurationen sind in produktspezifischen Implementierungen wie der ActiveMQ-Authentifizierung [5] zu sehen.

Hinweis: Es verfügt über optionale Header für die Client-Authentifizierung wie folgt [1],

- Login: Die Benutzerkennung, die zur Authentifizierung gegenüber einem gesicherten STOMP-Server verwendet wird.
- passcode: Das Passwort, das zur Authentifizierung gegenüber einem gesicherten STOMP-Server verwendet wird.

6.9 Physikalische Anforderungen

STOMP unterstützt eingeschränkte Geräte. Einige IoT-Entwicklungsplattform-Suiten unterstützen das STOMP-Protokoll, um eine verteilte Kommunikation zu ermöglichen. RoboMQ [6] ist beispielsweise eine Message Queue als Service-Plattform, die in der Cloud gehostet wird und auch als Enterprise Hosting-Option verfügbar ist. Es wurde für Cloud-, IoT- und heterogene Anwendungen entwickelt, die auch STOMP unterstützen.

6.10 Verfügbare Implementierungen

Es gibt verschiedene Open-Source-Implementierungen von STOMP [9]. Einige von ihnen sind Apache ActiveMQ, RabbitMQ, StompServer [9] für STOMP-Serverimplementierung und Activemessaging (Ruby), AnyEvent::Stomp (Perl), Apache CMS (C++) [9] für Client-Implementierung.

6.11 Referenzen

[1] "STOMP-Protokollspezifikation, Version 1.2." [Online]. Verfügbar unter: https://stomp.github.io/stomp-specification-1.2.html. [Zugriff: 08-Sep-2017].

[2] "Apache ActiveMQ." [Online]. Verfügbar unter: http://activemq.apache.org/stomp.html. [Zugriff: 15-Sep-2017].

[3] "Das WebSocket-Protokoll RFC 6455." [Online]. Verfügbar unter: https://tools.ietf.org/html/rfc6455. [Zugriff: 15-Sep-2017].

"Alles über Messaging-Protokolle." [Online]. Verfügbar unter:

http://www.eejournal.com/article/20150420-protocols/. [Zugriff: 11-Sep-2017].

[5] J. Mesnil, "Mobile und Web Messaging". [Online]. Verfügbar unter: https://www.safaribooksonline.com/library/view/mobile-and-web/9781491944790/ch04.html. [Zugriff: 11-Sep-2017].

[6] "IoT- und M2M-Integration." [Online]. Verfügbar unter: https://robomq.readthedocs.io/en/latest/deviceIntegration/#iot-and-m2m-integration. [Zugriff: 15-Sep-2017].

[7] "RabbitMQ STOMP Adapter." [Online]. Verfügbar unter: https://www.rabbitmq.com/stomp.html. [Zugriff: 15-Sep-2017].

[8] "RabbitMQ Web STOMP Plugin." [Online]. Verfügbar unter: https://www.rabbitmq.com/web-stomp.html. [Zugriff: 15-Sep-2017].

[9] "STOMP Server und Clients." [Online]. Verfügbar unter: https://stomp.github.io/implementations.html. [Zugriff: 08-Sep-2017].

7 Java Message Service (JMS)

7.1 Allgemeine Beschreibung

JMS ist eine API (Application Programming Interface) für den Zugriff auf Unternehmens-Messaging-Systeme aus Java-Programmen [1]. Das Ziel dieser API ist es, Java-Anwendungen mit den gemeinsamen Messaging-Konzepten/Funktionalitäten auszustatten, die für Enterprise Messaging erforderlich sind. Es gibt zwei Arten von Messaging-Stilen, die von JMS unterstützt werden, und zwar Punkt-zu-Punkt (PTP) und Publish/Subscribe (Pub/Sub) Methoden. Einer der Anwendungsfälle ist IBM MQ (Messaging Middleware) [4]. Die berücksichtigten Referenzen [10], [6] besagen, dass die Interoperabilität mit anderen Plattformen (außer JAVA) erheblich eingeschränkt ist. Es ermöglicht die Interoperabilität zwischen anderen Sprachen der Java-Plattform wie Scala und Groovy, bietet aber keinen Standard für die Interoperabilität außerhalb der Java-Plattform oder zwischen anderen Sprachen. Es zeigt, dass plattformübergreifende Interoperabilität zwar durchaus möglich ist, aber restriktiv (begrenzt) und herstellerabhängig ist. Daher ist JMS nicht die richtige Lösung für plattformübergreifende Interoperabilität.

7.2 Ebene

JMS-API-Schnittstellen werden von JMS-Anbietern auf Anwendungsebene implementiert. JMS definiert kein Drahtprotokoll für das Messaging [1]. Ein Drahtprotokoll wird definiert, wenn die Anwendung interagieren muss. Der Begriff "Drahtprotokoll" wird verwendet, um eine gebräuchliche Methode zur Darstellung von Informationen auf Anwendungsebene zu beschreiben [13].

7.3 Standardisierung

Die JMS-Version 2.0 ist die neueste verfügbare Spezifikation [1].

7.4 Architektur

Es handelt sich um eine nachrichtenorientierte Middleware.

7.5 Messaging-Paradigma

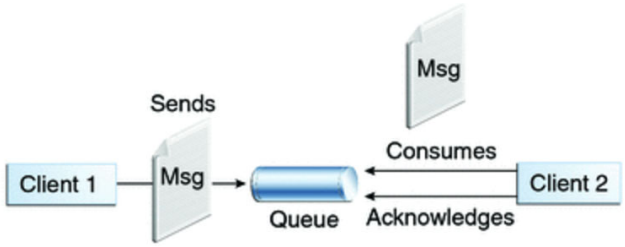

Abbildung 26 Punkt-zu-Punkt-Messaging [2].

PTP wird wie folgt erklärt [2],Abbildung 26 zeigt, dass PTP-Messaging es einem Client ermöglicht, eine Nachricht an einen anderen Client über eine Zwischenabstraktion namens Queue zu senden. Der Client, der die Nachricht sendet, sendet sie an eine bestimmte Warteschlange. Der Client, der die Nachricht erhält, extrahiert sie aus dieser Warteschlange [1].

Dieser Messaging-Stil kann verwendet werden, wenn jede gesendete Nachricht von einem Verbraucher erfolgreich verarbeitet werden muss [2]. Eine Warteschlange kann jedoch mehr als einen Verbraucher haben.

Abbildung 27 Publsih/Subscribe von Messaging [2].

Die Publish/Subscribe-Methode wird wie folgt erläutert, Abbildung 27 Abbildung 27zeigt Pub/Sub-Messaging. In einem Pub/Subprodukt oder einer Anwendung sprechen Clients Nachrichten zu einem Thema an. Publisher und Subscriber sind in der Regel anonym und können die Inhaltshierarchie dynamisch veröffentlichen oder abonnieren. Das System kümmert sich um die Verteilung der Nachrichten, die von den verschiedenen Publishern eines Topics eingehen, an seine mehreren Subscriber. Topics behalten Nachrichten nur so lange, wie es dauert, sie an aktuelle Subscriber zu verteilen [2]. Hinweis: Die Bestätigung der Nachricht wird in beiden Stilen verwendet.

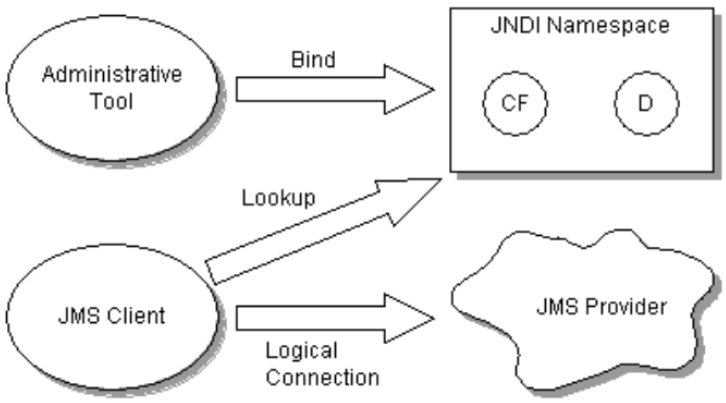

Abbildung 28 JMS-Verwaltung [1].

Abbildung 28 zeigt, wie die verwalteten Objekte wie Themen und Warteschlangen von den JMS-Clients entdeckt werden. Die JNDI (Java Naming and Directory Interface)[3] enthält zwei von JMS verwaltete Objekte, nämlich CF (ConnectionFactory) und D (Destination).

- ConnectionFactory - Dies ist das Objekt, mit dem ein Client eine Verbindung zu einem Provider herstellt.
- Ziel - Dies ist das Objekt, das ein Client verwendet, um das Ziel der gesendeten Nachrichten und die Quelle der empfangenen Nachrichten anzugeben. Hinweis: Das Ziel ist der gängige Supertyp des Themas oder der Warteschlange.

Hier erkennt der JMS-Client die verfügbaren Topics oder Warteschlangen über den JNDI-Namensraum. Und stellt dann die logische Verbindung zum Provider her.

7.6 Echtzeitfähigkeit

Das JMS unterstützt keine Echtzeitkommunikation [6], [7].

7.7 Servicequalität

Das JMS definiert eine Reihe von Zuverlässigkeitsattributen für Messaging, um es Entwicklern zu ermöglichen, die QoS-Anforderungen ihrer Anwendungen festzulegen, und sie sind in [9] wie folgt aufgeführt:

- Non-Persistent/Persistent (Zusicherung der Zustellung der ausstehenden Nachrichten, es handelt sich um Nachrichtenzustellarten)
- Kurzfristig/Dauerhaft (Empfangen von Nachrichten im aktiven oder inaktiven Zustand der Abonnenten)
- Nicht-transaktionale / transaktionale (lokale /verteilte Transaktionen)

"Die JMS-Spezifikation definiert den Satz der erforderlichen QoS-Attribute, nicht wie sie implementiert werden sollen, und so implementieren die Lieferanten diese Funktionalität auf unterschiedliche Weise, was zu unterschiedlichen Leistungsniveaus führt" [9].

Die folgende Tabelle zeigt die Zuverlässigkeit der gelieferten Nachricht (siehe Tabelle 4-1 in [1]), z.B. in einem NON PERSISTENT-Modus, wie eine Nachricht an einen nicht dauerhaften Teilnehmer veröffentlicht wird, der in der folgenden Tabelle angibt, dass eine Nachricht einmal veröffentlicht wird und verpasst wird, wenn der Teilnehmer inaktiv ist.

Wie veröffentlicht	Unbefristete Abonnenten	Langlebiger Abonnent
NICHT_PERSISTENTEN	höchstens einmal (verpasst bei Inaktivität)	höchstens einmal
PERSISTENT	genau einmal (verpasst bei Inaktivität)	genau einmal

Neues Feature der JMS 2.0 Spezifikation [1] (Erscheinungsdatum 20. März 2013),

"Lieferverzögerung: Ein Nachrichtenproduzent kann nun festlegen, dass eine Nachricht erst nach einem bestimmten Zeitintervall zugestellt werden darf" [1].

7.8 Sicherheit

Client-Authentifizierung: Es werden Benutzernamen/Passwort-Anmeldeinformationen verwendet (siehe Abschnitt 6.1.1 in [1]).

7.9 Physikalische Anforderungen

JMS kann für den Datenaustausch zwischen Low-Power-Geräten und Back-End-Infrastrukturen aufgrund der Herausforderungen an Interoperabilität und Sicherheit für die verteilte Kommunikation nicht in Betracht gezogen werden [6].

7.10 Verfügbare Implementierungen

Für die Open-Source-Implementierung von JMS [12] gibt es verschiedene Möglichkeiten. OpenJMS ist beispielsweise eine Open-Source-Implementierung der Java Message Service API 1.0.2 Spezifikation von Sun Microsystems. Es gibt verschiedene Funktionen, die von dieser speziellen Implementierung unterstützt werden.

7.11 Referenzen

[1] M. Hapner, R. Burridge, R. Sharma, J. Fialli, K. Stout und N. Deakin, "Java Message Service v2.0", nein. März 2013.

[2] "JAVA Message Service Konzept." [Online]. Verfügbar unter: https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpoq/index.html. [Zugriff: 07-Sep-2017].

[3] "Überblick über JNDI." [Online]. Verfügbar unter: https://docs.oracle.com/javase/tutorial/jndi/overview/index.html. [Zugriff: 07-Nov-2017].

[4] "IBM MQ." [Online]. Verfügbar unter: https://www-03.ibm.com/software/products/en/ibm-mq. [Zugriff: 30-Jan-2018].

[5] R. Monson-Haefel und D. A. Chappel, *Java Message Service*, 1. Auflage .

[6] V. M. Trifa, D. Gurnard, und M. Koehler, "Messaging-Methoden in einer serviceorientierten Architektur für industrielle Automatisierungssysteme", *Proc. INSS 2008 - 5th Int. Konf. Vernetzter Sensor, Syst.* S. 35-38, 2008.

[7] V. Gazis *et al.*, "A survey of technologies for the internet of things", *IWCMC 2015 - 11th Int. Wirel. Kommun. Mafia. Berechnung. Konf.* S. 1090-1095, 2015.

[8] A. (PrismTech) Foster, "Messaging Technologies for the Industrial Internet and the Internet of Things", nein. März, S. 1-22, 2014.

[9] P. Greenfield und S. Chen, "QoS-Evaluierung von JMS: ein empirischer Ansatz", *37. Jahrgang. Hawaii Int. Konf. Syst. Sci. 2004. Verfahren*, Bd. 0, Nr. C, S. 1-10, 2004.

[10] M. Richards, "Die Unterschiede zwischen AMQP und JMS verstehen", *No Fluff, Just St.*, Vol. 3, No. 3, S. 26-30, 2011.

[11] "Java Community Process." [Online]. Verfügbar unter: https://www.jcp.org/en/home/index. [Zugriff: 13-Nov-2017].

"Open Source JMS." [Online]. Verfügbar unter: https://java-source.net/open-source/jms. [Zugriff: 07-Sep-2017].

[13] "Drahtprotokoll"[Online]. Verfügbar: https://en.wikipedia.org/wiki/Wire_protocol. [Zugriff: 08-Aug-2018]

8 Message Queuing Telemetry Transport (MQTT)

8.1 Allgemeine Beschreibung

Der Message Queuing Telemetry Transport (MQTT) [1] ist ein leichtgewichtiges, Broker-basiertes Publish/Subscribe-Messaging-Protokoll und wurde von OASIS standardisiert. Es ist so konzipiert, dass es offen, einfach, einfach zu implementieren und für eingeschränkte Geräte mit begrenzten Verarbeitungs- und Speicherfähigkeiten geeignet ist, um Daten über Netzwerke mit niedriger Bandbreite zu senden. Das Protokoll ist für hochverzögerte oder unzuverlässige Netzwerke und für die Machine-to-Machine-Kommunikation im Internet der Dinge (IoT) ausgelegt. Die Designprinzipien von MQTT zielen darauf ab, den Bedarf an Netzwerkbandbreite und Geräteressourcen zu minimieren und gleichzeitig Zuverlässigkeit zu gewährleisten, indem drei Quality of Service Levels definiert werden.

8.2 Ebene

MQTT ist ein Application Layer Protocol [7] und hat Implementierungen in verschiedenen Programmiersprachen wie Java, C, C++, JavaScript, Lua, Python und bald auch C#. Diese ermöglichen eine einfache Bereitstellung auf bestehenden Geräten und die Integration in Unternehmenssysteme.

8.3 Standardisierung

MQTT wurde 2013 von der Organisation zur Förderung von strukturierten Informationsstandards (OASIS) standardisiert.

8.4 Architektur

MQTT ist eine nachrichtenorientierte Middleware.

Im Gegensatz zu High-Level-Konzepten wie SOA oder ROA ist ein MOM eine eigentliche Implementierung einer Middleware-Plattform. Es kann zur Implementierung einer SOA, einer EDA oder anderer Architekturen verwendet werden. Normalerweise bereichert ein MOM eine Reihe von Anwendungen mit asynchronem Messaging, bei dem ein MOM-Server die Nachrichten speichert und weiterleitet.

8.5 Messaging-Paradigma

MQTT basiert auf dem Publish/Subscribe-Paradigma. Sie besteht aus mindestens einem Publisher, einem Abonnenten und einem zentralen Broker. Der MQTT Broker ist ein Server, der alle Nachrichten von den Clients empfängt und sie dann an bestimmte Ziele weiterleitet. In einer typischen Publish/Subscribe-Architektur wie in Abbildung 29gibt es mehrere Publisher, die sich mit dem Broker verbinden, um ihre Daten zu senden, z.B. Geräte, die Sensordaten bereitstellen. Themen ermöglichen es Verlagen, ihre Daten zu kategorisieren. Die Teilnehmer könnten Anwendungen oder andere Geräte sein, die sich für bestimmte Themen interessieren und diese abonnieren können. Sie erhalten alle neuen Daten, die zu den abonnierten Themen veröffentlicht werden. Das Publish/Subscribe-Modell hat gegenüber dem Request/Response-Modell [2] Vorteile, da Push-Messaging in der Regel schneller ist und keine vorherigen Ressourcenanfragen erfordert.

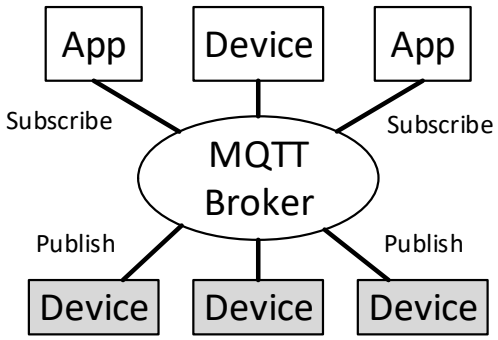

Abbildung 29 MQTT Veröffentlichungs-/Abonnementarchitektur

8.6 Echtzeitfähigkeit

MQTT ist darauf ausgerichtet, den Overhead zu minimieren und hochfrequentes Telemetrie-Messaging anzubieten. MQTT bietet Reaktionsfähigkeit, um eine sanfte Bereitstellung von Informationen in Echtzeit zu ermöglichen [11].

8.7 Servicequalität

MQTT definiert drei Quality of Service (QoS)-Stufen, um die Zuverlässigkeit der Nachrichtenzustellung zu gewährleisten:

QoS-Ebene 0:	Nachrichten werden höchstens einmal zugestellt. Es kann zu einem Nachrichtenverlust kommen.
QoS-Ebene 1:	Nachrichten werden mindestens einmal durch Acknowledgments (mit der Funktion PUBACK/SUBACK) zugestellt. Die gleiche Nachricht kann mehrmals empfangen werden.
QoS Level 2:	Nachrichten werden genau einmal zugestellt. In einem ersten Schritt sendet der Verlag einen PUBLISH, dann erhält er einen PUBACK, dann sendet er einen PUBREL und am Ende dieser Transaktion erhält er einen PUBCOMP. Nachrichten werden nicht mehrfach gesendet oder empfangen. QoS 2 könnte für den Geldverkehr eingesetzt werden.

MQTT bietet weitere QoS-Funktionen wie:

- "Letzter Wille und Testament": Wenn der Client unerwartet die Verbindung zum Broker trennt, wird die letzte Nachricht gespeichert, die der Client durch die Verbindung zum Broker angibt. Wenn der Broker feststellt, dass sich der Client abrupt trennt, sendet der Broker die Nachricht an alle abonnierten Kunden zu dem jeweiligen Thema.
- "Message Persistence": Wenn die Verbindung zwischen Broker und Client getrennt wird, kann der Broker eine neue Nachricht für diesen Client speichern, um sie später zu übertragen.
- "Heartbeat & Keep Alive": Clients können die Keep-Alive-Zeit definieren, die inaktive Verbindungen hält. Beim Senden einer PINGREQ kann der Client die Keep-Alive-Zeit auf Null setzen [3].

8.8 Sicherheit

MQTT bietet ein leichtes und einfach zu bedienendes Kommunikationsprotokoll, daher sind nicht viele Sicherheitsmerkmale spezifiziert. Die Idee von MQTT ist es, es durch anerkannte Standards zu ergänzen. Die Authentifizierung erfolgt mit der eindeutigen Client-Kennung, die der Client in der MQTT

CONNECT-Nachricht bereitstellt. Im Authentifizierungsprozess werden neben der Benutzername/Passwort-Authentifizierung auch Client-IDs verwendet. Dies ermöglicht es, den Zugriff des Kunden auf bestimmte Themen zu beschränken und die Veröffentlichung oder das Abonnement von nicht autorisierten Themen zu verhindern [4], [6]. MQTT kann mit TLS zur Transportverschlüsselung verwendet werden. TLS (Transport Layer Security) bietet einen sicheren Kommunikationskanal zwischen einem Client und einem Server [5].

8.9 Physikalische Anforderungen

Fixed header, present in all MQTT Control Packets
Variable header, present in some MQTT Control Packets
Payload, present in some MQTT Control Packets

Abbildung 30 Struktur eines MQTT Control Packets [1].

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

Abbildung 31 Festes Header-Format [1]

MQTT-Clients sind klein und können auf Geräten wie Sensoren und Aktoren laufen, die durch wenig Speicher, Rechenleistung und geringen Keep-Alive-Verkehr eingeschränkt sind. Es ist leicht und minimiert den Overhead, indem es die Kontrollpakete so klein wie möglich hält. Wie in Abbildung 30 zu sehen ist, geschieht dies durch Unterteilung in drei Teile, einen festen Kopf, einen variablen Kopf und eine Nutzlast. Der 2 Byte feste Header (Abbildung 31) ist immer vorhanden, während der größere variable Header und die Nutzlasten nur gelegentlich hinzugefügt werden.

Durch minimale Header, geringen Speicherbedarf und begrenzte Abhängigkeit von Bibliotheken ist MQTT speziell für eingeschränkte Geräte konzipiert.

8.10 Verfügbare Implementierungen

MQTT ist ein offener Standard. Kostenlose Versionen sind z.B. Eclipse Mosquitto mit C und C++ Client-Bibliotheken [9], HiveMQ[8] und Eclipse Paho [10].

8.11 Referenzen

MQTT Version 3.1.1.1 Plus Errata 01, Andrew Banks (IBM), Rahul Gubta (IBM), 2015

2] Rahul Gubta, IBM Global Technology Services, USA[Online]. Verfügbar unter: https://www.ibm.com/developerworks/community/blogs/5things/entry/5_things_to_know_about_mqtt_the_protocol_for_internet_of_things?lang=en[Zugriff: 08-Sep-2017].

3] IBM, WebSphere, "Aufbau von mobilen Echtzeit-Lösungen mit MQTT und IBM MessageSight",[Online]. Verfügbar unter: http://www.redbooks.ibm.com/redbooks/pdfs/sg248228.pdf[Zugriff: 07-Sep-2017].

Thomas Bayer, MQTT, Das M2M und IoT Protokoll, 04.03.2016[Online]. Verfügbar unter: https://www.predic8.de/mqtt.htm[Zugriff: 04-Sep-2017].

[5] S. Zamfir, T. Balan, I. Iliescu, F. Sandu, "A Security Analysis on Standard IoT Protocols", 2016.

6] IBM Knowledge Center, MQTT Sicherheit[Online]. Verfügbar: https://www.ibm.com/support/knowledgecenter/en/SS9D84_1.0.0/com.ibm.mm.tc.doc/tc00

150_.htm [Zugriff 07-Sep-2017]

[7] Beliebte Application Layer Protokolle, OSI Layer 7 - Application Layer, [Zugriff: 11-Sep-2017].

(8) Eclipse Mosquitto, (Online). Verfügbar unter: https://mosquitto.org/. [Zugriff: 21-Sep-2017].

HiveMQ Enterprise MQTT Broker, [Online]. Verfügbar unter http://www.hivemq.com/. [Zugriff 21-Sep-2017].

[10] Eclipse Paho, [Online]. Verfügbar unter: http://www.eclipse.org/paho/. [Zugriff 21-Sep-2017].

Gastón Carlos Hillar, "MQTT Essentials - A Lightweight IOT Protocol", 2017.

IBM Knowledge Center, "Von MQ Telemetry unterstützte Geräte"[Online]. Verfügbar unter https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.pro.doc/q003060_.htm[Zugriff 22.12.2017].

9 Data Distribution Service (DDS)

9.1 Allgemeine Beschreibung

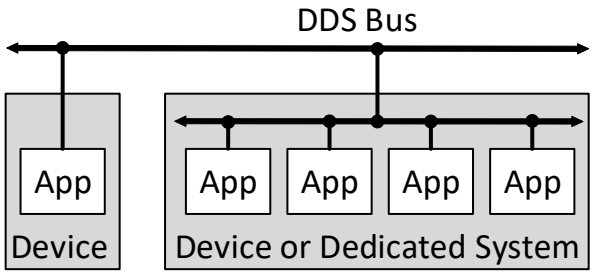

Abbildung 32 DDS-Architektur in einem Edge-Netzwerk mit Edge-Geräten

Der Data Distribution Service (DDS) [11] ist ein Datenkommunikationsmiddleware-Protokoll und API-Standard für datenzentrische Konnektivität, der von der Object Management Group (OMG) verwaltet wird. DDS bietet eine hohe Anzahl von Quality of Service (QoS)-Parametern, die sich auf die Drahtinteroperabilität, die geringe Latenzzeit der Datenverbindung, die Zuverlässigkeit und eine skalierbare Architektur beziehen. DDS-Implementierungen bieten Datenkommunikation, die für harte Echtzeit-Systeme geeignet ist. Durch die Reduzierung der Anzahl der Speicherzuweisungen [16] während der Laufzeit versucht DDS, seine Leistung zu verbessern. Der Datenverteilungsservice verwendet ein brokerloses Publish/Subscribe-Muster. Daher sind die Implementierungen nicht erforderlich, um einen zentralen Broker zur Verbindung von Publisher und Subscriber zu verwenden. Das Design bietet viele Vorteile, wie Fehlertoleranz und Skalierbarkeit. Darüber hinaus benötigen die Anwendungen nie Informationen über die andere teilnehmende Anwendung. Der Hauptzweck von DDS ist die Bereitstellung von Punkt-zu-Punkt-Verbindungen zwischen Geräten. Wie inAbbildung 32 zu sehen ist, arbeitet DDS in einem Edge-Netzwerk. Ein Edge-Netzwerk befindet sich an den logischen Extremen eines Netzwerks. Um die Kommunikationsbandbreite zwischen Sensoren und dem zentralen Rechenzentrum zu reduzieren, werden Rechenleistung und Dienste direkt in Edge-Geräte geleitet, die sich in diesen Edge-Netzen befinden. Anwendungsspezifische Datentypen, sogenannte Themen, werden auf den Geräten gehostet und Daten werden ohne den Einsatz eines Brokers veröffentlicht und untereinander abonniert. Ein Broker kann ein zentrales Element in einem Nachrichtenparadigma sein, das die empfangenen Nachrichten der Publisher verwaltet und an die jeweiligen Abonnenten sendet. Der datenzentrierte Middleware-Standard ist fernzugriffsfähig und kann Millionen von

Nachrichten pro Sekunde an viele gleichzeitige Teilnehmer veröffentlichen. DDS wird in Technologien wie industriellen Embedded-Anwendungen, militärischen Anwendungen und der Weltraumforschung eingesetzt [1].

Die DDS-Spezifikation ist in mehrere Teile gegliedert und beschreibt die bereitgestellten Dienste in der Unified Modeling Language (UML) auf Basis des Platform Independent Model (PIM). Das PIM gewährleistet die Portabilität von Implementierungen in jeder Programmiersprache und auf jedem Betriebssystem. Die DDS Core Spezifikationen beinhalten die Data-Centric Publish-Subscribe (DCPS)-Schicht und das Real-Time Publish-Subscribe (RTPS)-Protokoll. Das DCPS spezifiziert die Datenverteilungsarchitektur, während RTPS das Drahtprotokoll ist, das für den Transport der Daten verantwortlich ist.

9.2 Ebene

Der Datenverteilungsdienst arbeitet auf der Anwendungsebene. Das DDS Real-Time Publish-Subscribe (RTPS) Kabelprotokoll bietet Interoperabilität zwischen Implementierungen, Plattformen und Programmiersprachen wie C, C++, C# und Java. Es ist polyglott und plattformunabhängig [9], [10].

9.3 Standardisierung

DDS wurde von der OMG [10] standardisiert und ist in vielen großen Anwendungen, die eine verteilte Kommunikation erfordern, weit verbreitet, einschließlich Gesundheitswesen, Militär, Industrie und öffentliche Infrastrukturen.

9.4 Architektur

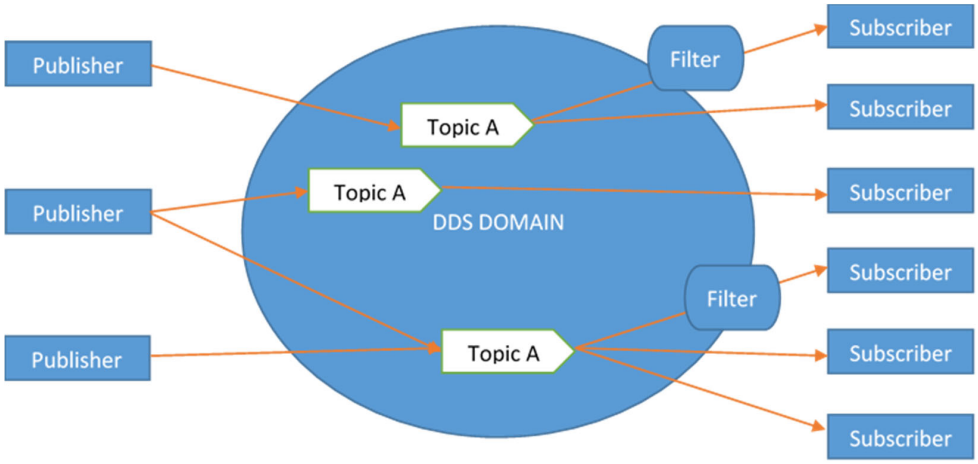

Abbildung 33 Eine DDS-Domäne im globalen Datenraum

Die DDS-Architektur ist dezentralisiert, um eine hohe Zuverlässigkeit und Datenkonnektivität mit geringer Latenz für geschäftskritische IoT-Anwendungen zu gewährleisten. Daten in DDS werden in einem globalen Datenraum aus DDS-Domänen bereitgestellt (Abbildung 33). Einzelne Anwendungen speichern ihre erforderlichen Daten lokal zwischen, während der Rest in entfernten Knoten gehalten und über APIs wie im nativen Speicher der Anwendung abgerufen wird. Basierend auf den Prinzipien des Publish/Subscribe-Paradigmas werden Daten gelesen oder in Themen geschrieben. DDS-Domänen teilen den Datenraum logisch auf, um die Kommunikation und Adressierung zu optimieren. Teilnehmer können sich zu Themen innerhalb einer Domain anmelden und erhalten jede Nachricht, die von anderen Teilnehmern zu diesem Thema veröffentlicht wird.

Aus Sicht einer Anwendung hat sie direkten lokalen Zugriff auf den gesamten globalen Datenraum, während in Wirklichkeit relevante Daten bei Bedarf aus der Ferne abgerufen werden. Der globale

Datenraum ist somit eine Sammlung von lokalen Anwendungsspeichern, die allen Teilnehmern auf Peer-to-Peer-Basis zur Verfügung stehen. Alle Geräte sind im Wesentlichen an einen DDS-Bus angeschlossen (Abbildung 32) und kommunizieren miteinander, ohne dass eine Cloud oder ein Serverbroker benötigt wird. Diese dezentrale Architektur ermöglicht es Systemen, in Echtzeit zu kommunizieren und gleichzeitig eine große Anzahl von Teilnehmern zu skalieren, von Edge bis hin zu Cloud-Netzwerken.

DDS bietet eine dynamische Ermittlung von Publishern und Abonnenten. Die dynamische Erkennung macht DDS-Anwendungen erweiterbar, die Endpunkte für die Kommunikation werden automatisch erkannt und konfiguriert [3]. Einzelne DDS-Instanzen können als lose gekoppelte interoperable Dienste betrachtet werden, die eine serviceorientierte Architektur bilden. Aufgrund der Art des verwendeten Publish/Subscribe-Paradigmas und des Fokus auf Daten und Datenbeziehungen, die das Gefüge von Geschäftsprozessen bilden, kann DDS jedoch als ereignisgesteuerte Architektur betrachtet werden.

9.5 Messaging-Paradigma

Der Data Distribution Service verwendet in erster Linie ein Publish/Subscribe-Modell und bietet QoS-gesteuerten Datenaustausch. Die Anwendungen kommunizieren, indem sie Themen veröffentlichen und abonnieren, die durch ihren Themennamen identifiziert werden. Die Publisher tragen Datenobjekte bei, und die Abonnenten melden den Zugriffswert. Die Verlage erklären die Themen, zu denen sie ihre Daten veröffentlichen wollen, und die Datenkonsumenten abonnieren die Themen von Interesse. Die Abonnenten können Zeit- und Inhaltsfilter für den Datenempfang festlegen. Alle Abonnenten, die ein bestimmtes Thema abonniert haben, erhalten die Daten bei der Veröffentlichung. Der Globale Datenraum identifiziert Daten, die im System zirkulieren, und bietet Datenisolierungsmechanismen zur Verbesserung der System-Skalierbarkeit. Das Data-Centric Publish/Subscribe (DCPS) ist eine Low-Level-Schicht, die eine effiziente Bereitstellung von Daten ermöglicht, die von den Publishern an die Abonnenten weitergegeben werden [5, [6].

9.6 Echtzeitfähigkeit

DDS bietet harte Echtzeitkommunikation. Es ermöglicht einen vorhersehbaren und deterministischen Datenaustausch zwischen Anwendungen, indem es den Benutzern die Kontrolle über das Echtzeitverhalten des Systems ermöglicht. QoS kann spezifiziert werden, um Timing, Priorität des Kommunikationskanals und Ressourcenauslastung zu steuern [13].

9.7 Servicequalität

DDS bietet einen umfangreichen Satz von Quality of Service (QoS)-Richtlinien, um das Verhalten der Kommunikation anzupassen. Über zwanzig QoS-Richtlinien können einzeln oder gemeinsam verwendet werden, um eine Vielzahl von Kommunikationsaspekten zu beeinflussen, einschließlich Zuverlässigkeit, Leistung, Datenpersistenz und Sicherheit [4]. Die Stärke von DDS liegt darin, all diese Funktionalitäten gleichzeitig, bei extrem hohen Geschwindigkeiten und in sehr dynamischen, anspruchsvollen und unvorhersehbaren Umgebungen bereitzustellen [2], [3].

9.8 Sicherheit

DDS bietet standardisierte Authentifizierungs-, Verschlüsselungs-, Zugriffskontroll- und Protokollierungsfunktionen, um eine sichere End-to-End-Datenverbindung in einem verteilten System zu ermöglichen [7]. Die Sicherung von DDS bedeutet die Bereitstellung von:

- Vertraulichkeit der Datenproben
- Integrität der Datenproben und der Nachrichten, die sie enthalten
- Authentifizierung von DDS-Schreibern und Lesern
- Autorisierung von DDS-Schreibern und Lesern

- Unleugbarkeit der Daten

Alle kommunizierenden Parteien, einschließlich Geräte und Benutzer, werden separat authentifiziert. Die Implementierung der Zugriffskontrolle ermöglicht es, die Veröffentlichung oder das Abonnement bestimmter Themen im DDS-Netzwerk zu erlauben oder zu verhindern. Die Sicherheitsimplementierungen in DDS befinden sich oberhalb der Transportschicht, TLS/SSL oder DTLS sind nicht erforderlich. Die Verwendung von TLS würde eine Echtzeit- und Multicast-Kommunikation verhindern. Die Authentifizierungsimplementierung wird verwendet, um alle Teilnehmer wie Benutzer und Geräte zu authentifizieren. Die Implementierung der Zugriffskontrolle ermöglicht es der Steuerung, die Veröffentlichung oder das Abonnement bestimmter Arten von Daten über das DDS-Netzwerk zu ermöglichen. Die Verschlüsselung von Daten, die sich über das DDS-Netzwerk bewegen, ist wichtig, erhöht aber Bandbreite und Overhead, so dass der Entwickler die Möglichkeit hat, die zu verschlüsselnden Daten auszuwählen oder nicht. [8].

9.9 Physikalische Anforderungen

DDS hat einen sehr kleinen Speicher- und Bandbreitenbedarf und wurde entwickelt, um eine Echtzeitkommunikation zwischen Geräten zu ermöglichen. Es gibt Implementierungen wie Vortex Lite Webcast, die sich ganz auf die Optimierung von DDS für Embedded-Umgebungen konzentrieren [14].

9.10 Verfügbare Implementierungen

DDS ist ein offener Standard. Es gibt verschiedene Open-Source- und kommerzielle proprietäre Implementierungen [12]. Während die Implementierung der DCPS-Architektur frei verfügbar ist, ist die DDS Security-Implementierung zum jetzigen Zeitpunkt nur kommerziell verfügbar.

9.11 Referenzen

[1] Alasdair Gilchrist, Industrie 4.0: Das industrielle Internet der Dinge, 2016.

2] OpenDDS, Nutzung der QoS-Richtlinien[Online]. Verfügbar unter: http://opendds.org/about/qosusages.htm[Zugriff: 14-Sep-2017].

3] OMG, DDS: Der bewährte Datenkonnektivitätsstandard für IoT, "Was ist DDS?". [Online]. Verfügbar unter: http://portals.omg.org/dds/what-is-dds-3/[Zugriff: 12-Sep-2017].

[4] Prismtech, Was sind die Vorteile von DDS? [Online]. Verfügbar unter: http://www.prismtech.com/ vortex/technologies/what-are-benefits-dds[Zugriff: 07-Sep-2017].

Gerardo Pardo-Castellote, Bert Farabaugh, Rick Warren, "An Introduction to DDS and Data-Centric Communications", 2005.

Paolo Bellavista, Antonio Corradi, Luca Foschini, Alessandro Pernafini, "Data Distribution Service (DDS)": Ein Leistungsvergleich von OpenSplice und RTI-Implementierungen", 2013.

7] DDS, Der bewährte Datenkonnektivitätsstandard für IoT, "Technische Vorteile"[Online]. Verfügbar unter: http://portals.omg.org/dds/key-technical-benefits/[Zugriff: 05-Sep-2017].

8] OMG, DDS Security Version 1.0, Sep-2016.

[9] TWINOAKS COMPUTING, "Was kann DDS für Sie tun?" [Online]. Verfügbar: http://www.omg.org/hot-topics/documents/dds/CoreDX_DDS_Why_Use_DDS.pdf[Zugriff: 4-Sep-2017].

[10] RTI, DDS Standard, [Online]. Verfügbar unter: https://www.rti.com/products/dds/omg-dds-standard. [Zugriff: 4-Sep-2017].

[11] OMG, Datenverteilungsdienst Version 1.4, April-2015.

12] OMG, DDS: Der bewährte Datenkonnektivitätsstandard für das IoT, "Where Can I Get DDS",[Online]. Verfügbar unter: http://portals.omg.org/dds/where-can-i-get-dds/[Zugriff: 08-Sep-2017].

[13] OMG, DDS: Der bewährte Datenkonnektivitätsstandard für das IotT, "Warum DDS?",[Online]. Verfügbar unter: http://portals.omg.org/dds/why-choose-dds/[Zugriff: 08-Sep-2017].

[14] PrismTech, "Vortex Lite Webcast vorstellen",[Online]. Verfügbar unter: https://blog.prismtech.com/2015/01/21/introducing-vortex-lite-webcast/[Zugriff: 24-Sep-2017].

[15] OMG, DDS: Der bewährte Datenkonnektivitätsstandard für das IoT, "Architektur",[Online]. Verfügbar unter: http://opendds.org/documents/architecture.html[Zugriff: 13-Nov-2017].

[16] Twin Oaks Computing, Inc., "Was kann DDS für Sie tun?",[Online]. Verfügbar unter: http://www.omg.org/hot-topics/documents/dds/CoreDX_DDS_Why_Use_DDS.pdf[Zugriff: 22.01.2018].

10 Advanced Message Queuing Protocol (AMQP)

10.1 Allgemeine Beschreibung

Das Advanced Message Queuing Protocol (AMQP) [1] ist ein offener Standard der AMQP-Arbeitsgruppe zur Übermittlung von Geschäftsnachrichten zwischen Anwendungen und Prozessen. Es verbindet Systeme und speist Geschäftsprozesse mit den Informationen, die sie benötigen. AMQP läuft über TCP und wurde entwickelt, um das Problem der Nachrichteninteroperabilität zwischen heterogenen Plattformen und Message Brokern in der Finanzindustrie zu lösen und Sicherheit und Zuverlässigkeit zu gewährleisten. Aufgrund des Einsatzes im Finanzumfeld muss AMQP die Erwartungen an Geschwindigkeit, Performance, Skalierbarkeit und Zuverlässigkeit erfüllen. Einer der Hauptziele von AMQP ist es, Nachrichten zwischen Anwendungen bereitzustellen und über ein einfaches Warteschlangensystem zu verfügen, um asynchrone Nachrichten über viele serverseitige Anwendungen zu erhalten [2].

10.2 Ebene

AMQP arbeitet auf der Applikationsebene und ist plattformunabhängig.

10.3 Standardisierung

Ziel der Entwicklung war es, eine Vielzahl unterschiedlicher Anwendungen und Systeme in die Lage zu versetzen, unabhängig von ihren internen Konstruktionen zusammenzuarbeiten und Unternehmensnachrichten im industriellen Maßstab zu standardisieren [4]. Sie wird von der Organisation zur Förderung von Standards für strukturierte Informationen (OASIS) festgelegt.

10.4 Architektur

AMQP ist eine Nachrichtenorientierte Middleware.

Im Gegensatz zu High-Level-Konzepten wie SOA oder ROA ist ein MOM eine eigentliche Implementierung einer Middleware-Plattform. Es kann zur Implementierung einer SOA, einer EDA oder anderer Architekturen verwendet werden. Normalerweise bereichert ein MOM eine Reihe von Anwendungen mit asynchronem Messaging, bei dem ein MOM-Server die Nachrichten speichert und weiterleitet.

10.5 Messaging-Paradigma

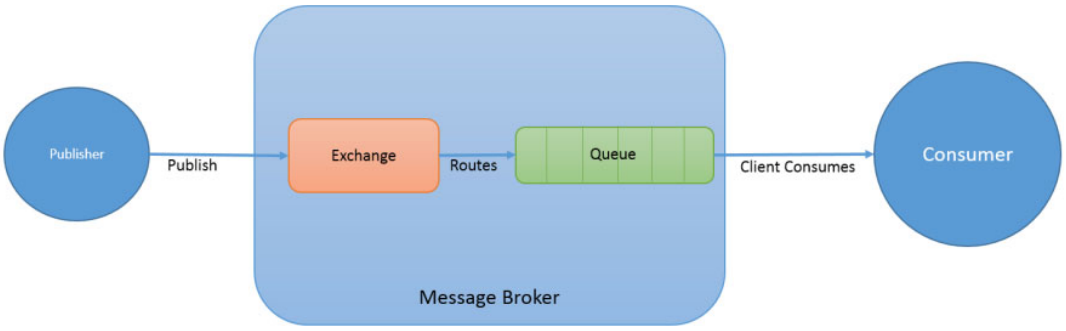

Abbildung 34 AMQP-Nachrichtenparadigma [9].

AMQP verwendet Punkt-zu-Punkt- und Publish-/Subscirbesysteme. Punkt-zu-Punkt stellt eine Nachrichtenwarteschlange zwischen zwei Kommunikationsendpunkten her (Abbildung 34). Publish/Subcribe ist ein Nachrichtenmuster, bei dem sich die Publisher, die Nachrichten senden, und die Subscriber, die Nachrichten empfangen, nicht kennen, was bedeutet, dass sie nur wissen, welche Art von Nachricht gesendet wird, aber nicht von wem oder an wen. Es kann mehrere Publisher und/oder Subsciber geben. AMQP muss sich mit realistischen Problemen von Unternehmensanwendungen befassen. Das bedeutet Handhabung von:

- Store-and-Forward-Messaging, ähnlich wie bei E-Mails.
- Veröffentlichen und Abonnieren von Nachrichten, ähnlich wie bei Newsgroups.
- Multicasting, bei dem Nachrichten parallel an viele Parteien gesendet werden.

AMQP Publisher senden keine Nachrichten direkt an Warteschlangen. Die Nachrichten werden zunächst an einen Exchanger veröffentlicht. Exchanger empfangen Nachrichten von dem Publisher und bringen sie in die Warteschlange. Routen definieren, auf welcher Warteschlange die Nachricht gesendet oder die Nachricht verworfen werden soll. Clients, die diese Warteschlange abonnieren, erhalten dann eine Kopie der Nachricht. Wenn mehr als ein Client dieselbe Warteschlange abonniert, werden die Nachrichten in einer Round-Robin-Form ausgegeben. Eine Nachricht wird an einen Exchange veröffentlicht, der durch seine Verknüpfungen an eine oder mehrere Warteschlangen gesendet werden kann. AMQP hat Broker, die für den Empfang, die Weiterleitung und die Ausgabe von Nachrichten an Clients verantwortlich sind [3].

10.6 Echtzeitfähigkeit

AMQP hat seinen Fokus auf QoS und zuverlässige Nachrichtenzustellung. Eine Echtzeitkommunikation ist nicht möglich.

10.7 Servicequalität

In AMQP steuert die QoS, wie schnell Nachrichten gesendet werden und hängt von der Art der verteilten Inhalte ab. Es gibt verschiedene QoS-Semantik, die für grundlegendes Messaging, Dateiübertragung und Streaming definiert ist. Die QoS gibt an, wie viele Nachrichten gesendet werden, bevor der Client eine Nachricht bestätigen muss. Die Nachrichtendaten werden daher vom Empfänger vorab abgerufen, um die Latenzzeit zu reduzieren. Die verschiedenen QoS-Optionen helfen, die Dienste auf einem bestimmten Leistungsniveau zu halten, wozu auch die Verarbeitung großer Nachrichten, die Gewährleistung maximaler Lieferzeiten und der Bandbreitenverbrauch für bestimmte Dienste gehört, die Übertragungsfehler auf robuste und nützliche Weise behandeln [6]. Bestätigungen ermöglichen es Clients, anzuzeigen, dass eine Nachricht empfangen und verarbeitet wurde.

Nachrichten werden erst dann bestätigt, wenn sie vollständig empfangen und bearbeitet wurden. Die Semantik der Nachrichtenwarteschlange umfasst gut ausgearbeitete Bereitstellungsmechanismen:

- At-mose-once: Das Paket kam einmal oder gar nicht an.
- At-least-once: Das Paket kam ein- oder mehrmals als Duplikat an.
- Exactly-once: Das Paket kam einmal ohne Duplikate an.

10.8 Sicherheit

Die AMQP-Spezifikation erwartet, dass die Sicherheitsschicht wie bei TLS extern definiert wird. AMQP bietet jedoch zusätzlich das notwendige Framing für den Simple Authentication Security Layer (SASL) [5]. SASL ist ein Framework, das die Unterstützung von Pluggable-Authentifizierung für bestehende Anwendungsprotokolle ermöglicht. Es gibt drei eingebaute Mechanismen:

- PLAIN: SASL PLAIN-Authentifizierung ist der Standard-SASL-Standardmechanismus für die Authentifizierung und bietet keine Datenverschlüsselung.
- AMQPLAIN: Eine Nicht-Standard-Version von PLAIN, die in AMQP 0-8 spezifiziert ist.
- RABBIT-CR-DEMO: Sicherheit entspricht PLAIN und zeigt eine Challenge-Response-Authentifizierung.
- EXTERN**:** Out-of-Band-Mechanismen, die von Plugins bereitgestellt werden. Z.B. X.509-Zertifikat PKI-Unterstützung für die Verwendung mit TLS.

10.9 Physikalische Anforderungen

AMQP wurde nicht für eingebettete Geräte entwickelt, da der Broker, der die Nachrichtenwarteschlange hostet, eine Vielzahl von QoS-Funktionen bereitstellt, die hohe Rechenressourcen erfordern und aufgrund großer Nachrichtengrößen einen Overhead verursachen.

10.10 Verfügbare Implementierungen

AMQP ist ein offenes Standardprotokoll. Es gibt mehrere Anbieter für Open Source mit unterschiedlicher Funktionalität, z.B. ZeroMQ und RabbitMQ [7], [8].

10.11 Referenzen

[1] OASIS Standard, OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0, 29. Oktober 2012.

[2] Mark Richards, "Understanding the Differences between AMQP & JMS", 2011.

[3] Andreas Haller, "Message Queuing Systems", Universität zu Lübeck, 2013.

[4] Joel L. Fernandes, Ivo C. Lopes, Joel J.P.C. Rodrigues, Sana Ullah, "Performance Evaluation of RESTful Web Services and AMQP Protocol", 2013.

[5] OpenLDAP Foundation, "Simple Authentication and Security Layer (SASL)", 2006.

[6] RabbitMQ, AMQP: Advanced Message Queuing Protocol, Protocol Specification,[Online]. Verfügbar unter: https://www.rabbitmq.com/resources/specs/amqp0-8.pdf[Zugriff: 15-Sep-2017].

[7] RabbitMQ, Verfügbar: https://www.rabbitmq.com/[Zugriff: 22-Sep-2017].

[8] ZeroMQ, verfügbar: http://zeromq.org/ [Zugriff: 22-Sep-2017].

9] Stephen Haunts, "RabbitMQ Serie Teil 2: Der AMQP Messaging-Standard",[Online]. Verfügbar unter: https://stephenhaunts.com/2015/09/09/rabbitmq-series-part-2-the-amqp-messaging-standard/[Zugriff: 18-Dez-2017].

11 Web Services

11.1 Allgemeine Beschreibung

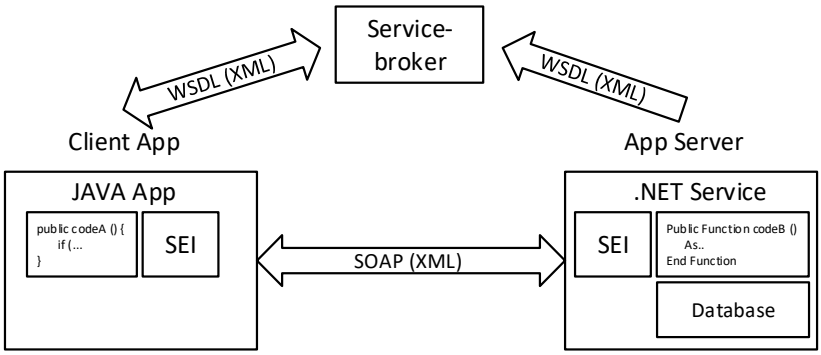

Abbildung 35 Allgemeines SOAP/WSDL-Modell mit einem Client und einem Server mit verschiedenen Technologien

Webdienste (WS) sind Dienste, die von Geräten für andere elektronische Geräte durch Kommunikation über lokale Netzwerke oder das World Wide Web angeboten werden. Softwareteile werden über Netzwerke bereitgestellt und dienen dem Austausch von Dokumenten oder der Bereitstellung von Funktionen als Remote Procedure Calls (RPCs) für die Machine-to-Machine-Kommunikation. Sie sind modular und in sich geschlossen, da die Hauptidee von WS darin besteht, eine Schnittstelle zu bestimmten Funktionen bereitzustellen, ohne den Rest der Systemarchitektur berücksichtigen zu müssen. Dies bedeutet auch, dass Änderungen innerhalb der Schnittstelle die Interaktionsfähigkeit eines Clients mit dem WS nicht beeinträchtigen und somit eine hohe Flexibilität und Skalierbarkeit ermöglichen.

Es gibt verschiedene Möglichkeiten, wie WS realisiert werden kann. Traditionell wird bei WS-Implementierungen das Simple Object Access Protocol (SOAP) verwendet, aber es besteht auch die Möglichkeit einer vereinfachten Realisierung, die ausschließlich auf dem Designkonzept des Representational State Transfer (REST) basiert.

SOAP WS

SOAP ist ein Datenformatprotokoll, das Anwendungsschichtprotokolle wie HTTP oder SMTP oder einfache binäre Verschlüsselung über TCP oder UDP als Transport verwendet. SOAP legt fest, wie Daten für den Austausch innerhalb der XML-Nachrichtenstruktur formatiert werden. Es basiert auf den WS-* Standards [1] [2] [3]. *WS-** ist eine gebräuchliche Abkürzung, um auf die Vielfalt der verschiedenen Spezifikationen für Web-Service-Messaging hinzuweisen. Sie beschreiben das grundlegende Web Service Framework, das teilweise in den SOAP/WSDL-Standards abgebildet ist. Nicht alle Spezifikationen werden in jeder WS-Implementierung verwendet, da es keinen festen Satz von Spezifikationen gibt. Einige Funktionen überschneiden sich oder konkurrieren und das WS-Präfix ist daher nur ein allgemeiner Hinweis auf Spezifikationen, die mit WS verbunden sind. Wie inAbbildung 35 zu sehen ist, verwenden beide kommunizierenden Parteien in SOAP ein Service Endpoint Interface (SEI), das Methodenaufrufe und Dokumente in SOAP/XML-Nachrichten und umgekehrt übersetzt. Auf diese Weise können Maschinen mit unterschiedlichen Technologien über das gemeinsame SOAP-Nachrichtenformat kommunizieren. Bevor die Kommunikation beginnen kann, müssen die WS-Attribute, Parameter und das Nachrichtenformat erfasst werden. Dies wird durch einen Service-Broker realisiert, der jedem interessierten Client die Datei Web Service Description Language (WSDL) für einen WS zur Verfügung stellt. Damit weiß der Client dann, wie er richtig auf den WS zugreifen kann und wie

er seine Methodenaufrufe oder Dokumente strukturiert. Die Nachrichten werden dann mit einem SOAP-Umschlag verpackt, der einen XML-Header und -Body enthält und zwischen den Maschinen ausgetauscht wird.

REST WS

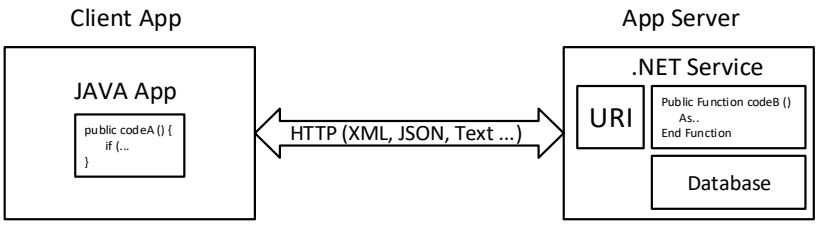

Abbildung 36 Allgemeines REST-Modell mit einem Client und einem Server mit verschiedenen Technologien

REST [4] ist keine Spezifikation, sondern ein Architekturstil zur Verwaltung von Zustandsinformationen in vernetzten Systemen. Es stellt öffentliche APIs über ein Netzwerk zur Verfügung, um CRUD-Operationen (Create, Read Update, Delete) auf Daten zu verarbeiten. Die Grundidee ist es, jede Ressource über einen Uniform Resource Identifier (URI) verfügbar zu machen. Die Darstellung der Ressourcen muss zustandslos und über eine einzige einheitliche Schnittstelle zugänglich sein. Daher müssen die Zustände der Anfragen jedes Clients an einen Server vollständig dargestellt werden. Der Server sollte in der Lage sein, die Client-Anfrage vollständig zu verstehen, ohne sich auf zusätzliche Informationen zur Serversitzung verlassen zu müssen. Wie inAbbildung 36 zu sehen ist, ist REST im Gegensatz zu SOAP nicht auf XML oder ein anderes spezifisches Protokoll beschränkt. Es ist jedoch eng an HTTP gebunden, da sie beide gemeinsam entwickelt wurden. REST beschreibt im Wesentlichen die Regeln, wie HTTP verwendet werden soll. Aufgrund der Einfachheit, Popularität und natürlichen Zuordnung von HTTP zu den allgemeinen RESTful-Prinzipien kann eine REST-API von praktisch jeder Programmiersprache aus aufgerufen werden und lässt sich leicht in jede MES/ERP-Lösung integrieren.

DPWS

Basierend auf den WS-*-Spezifikationen mit SOAP-, WSDL-, XML-, HTTP- und anderen WS-Standards identifiziert Device Profile for Web Services (DPWS) [19] einen Kernsatz von WS-Spezifikationen, um die Interoperabilität zwischen ressourcenbeschränkten WS und flexibleren Client-Implementierungen zu fördern. Das Ziel war es, sicheres Messaging zwischen WS, dynamische Erkennung und Beschreibung von WS und Abonnement-/Empfangsereignisse von WS bereitzustellen. Um WS-Implementierungen bereitzustellen, die mit dieser Mechanik konform sind, definiert der DPWS die Mindestanforderungen für die Implementierung für jeden dieser Bereiche. Dadurch können DPWS-fähige Geräte automatisch andere DPWS-Geräte im Netzwerk erkennen und über WSDL auf ihre WS-Beschreibungen zugreifen, damit Clients ihre Dienste per Plug-and-Play aufrufen können.

11.2 Ebene

SOAP arbeitet auf der Anwendungsschicht und bietet ein plattformunabhängiges Middleware-Messaging-Protokoll. Es gibt eine Vielzahl von Software-Frameworks, die SOAP-APIs bereitstellen. Diese können integriert und von Anwendungen aufgerufen werden, die in verschiedenen Sprachen geschrieben sind und auf unterschiedlichen Hardwarearchitekturen laufen.

REST WS werden im Allgemeinen über HTTP ausgeführt und können mit allen gängigen Nachrichtenaustauschformaten verwendet werden, wobei die beliebtesten XML- und JavaScript-Objektnotation (JSON) sind. Beide Formate sind gut etabliert und werden voraussichtlich von jeder modernen API unterstützt.

11.3 Standardisierung

SOAP wurde 2007 standardisiert [12] und ist ein weit verbreiteter Ansatz zur Realisierung von Web Services.

Das Internet, wie wir es kennen, arbeitet nach dem REST-Prinzip, da HTTP die treibende Kraft hinter dem Web ist und auf dem REST-Regelsatz basiert. RESTful WS kann in jedem Software-Framework realisiert werden und aufgrund ihrer Einfachheit haben die meisten großen Webservice-Provider wie Google oder Amazon begonnen, ihre Integrationsarchitektur auf REST umzustellen und sich von SOAP [13] zu entfernen.

11.4 Architektur

Beide Ansätze bieten von Natur aus Webservices an, wobei SOAP enger mit einer serviceorientierten Architektur (SOA) verbunden ist, während REST eher ressourcenorientiert ist. Der Unterschied zwischen SOA und ressourcenorientierten Architekturen (ROA) liegt vor allem in der Art und Weise, wie Prozesse gehandhabt werden und Daten "hinter den Bildschirmen" des Benutzers erfasst werden. In der SOA kombiniert oder komponiert eine Anwendung ihre dem Benutzer präsentierte Funktionalität auf der Grundlage verschiedener extern genannter Dienste. Informationen aus den Diensten werden gesammelt, indem man sie mit den erforderlichen Parametern versorgt und die Ergebnisse für die Benutzer kombiniert. Die Verarbeitung erfolgt überwiegend durch die externen Dienstleister. In ROA übernehmen Anwendungen den größten Teil der Verarbeitung selbst, verwenden aber externe Ressourcen als Dateneingabe. Ressourcen sind Softwarekomponenten wie Datenstrukturen oder diskreter Code. Informationen werden als statische Ressourcen von externen Dienstleistern abgerufen und die Kalkulation intern durchgeführt. Angenommen, ein WS liefert Ihnen die Information, ob Sie die Internationale Raumstation in einer bestimmten Nacht vorbeiziehen sehen. Mit einer SOA würde die Anwendung einen externen Dienst mit der Stadt und dem Datum als Parameter abfragen und eine Wahrscheinlichkeit erhalten, die ISS als Rückkehr zu sehen, basierend auf Standort- und Wetterberichten, die der Dienst aus seinen eigenen oder anderen externen Datenbanken erhalten hat. In einer ROA würde die Anwendung eine Ressource abrufen, die den ISS-Orbit während des angeforderten Zeitraums repräsentiert, und eine Ressource, die die Wettervorhersage für die Stadt enthält. Die Wahrscheinlichkeit, die ISS zu sehen, wird dann berechnet und dem Benutzer anhand der erfassten Daten präsentiert.

REST bietet daher im Allgemeinen eine höhere Flexibilität in Bezug auf die WS-Implementierung, da es sich nur auf zustandslose Darstellungen von Ressourcen stützt, die beliebig kombiniert werden können. Mit SOAP sind Sie auf die implementierten verfügbaren Dienste beschränkt, die mit der bereitgestellten WSDL-Datei beschrieben werden.

11.5 Messaging-Paradigma

Beide Ansätze basieren auf dem Request/Response-Modell, bei dem ein Client einen Server nach neuen Nachrichten fragen muss.

11.6 Echtzeitfähigkeit

SOAP benötigt einen XML-Parser, der die ausgetauschten Nachrichten während der Laufzeit interpretiert. Dies führt zu einer erhöhten Verarbeitungszeit auf den Endgeräten. Darüber hinaus sind XML und HTTP beide schwere Protokolle mit umfangreichen Nachrichtenstrukturen, die nicht für eingeschränkte Geräte entwickelt wurden. Bei kleinen Messages kann die SOAP-Hülle allein größer sein als die verfügbare Nutzlast. Alternativen wie das Constrained Application Protocol (CoAP) zum Beispiel (funktioniert wie HTTP, ist aber speziell für ressourcenbeschränkte Geräte konzipiert) erfordern bis zu 7 mal weniger Overhead als HTTP [10].

REST, wie auch SOAP, basiert auf dem Anfrage-/Antwortmodell, bei dem Informationen durch Umfragen gesammelt werden. Dies führt zu zusätzlichen Verzögerungen. Ähnlich wie bei SOAP führt die Verwendung von HTTP in REST-Anwendungen aufgrund der recht großen Kopf- und Nachrichtenstruktur von HTTP zu hohen Overheads. Beide Ansätze sind daher nicht für die Echtzeitkommunikation geeignet.

11.7 Servicequalität

SOAP bietet verschiedene QoS-Funktionalitäten, die in den WS-*-Spezifikationen beschrieben sind. WS-ReliableMessaging [11] ermöglicht die zuverlässige Zustellung von SOAP-Nachrichten bei Ausfällen während der Kommunikation.

REST stellt keine Mittel für zusätzliche QoS zur Verfügung. Diese müssen von den kommunizierenden Anwendungen selbst angegangen werden.

11.8 Sicherheit

SOAP kann das WS-Security Framework für End-to-End-Sicherheit nutzen [14]. Es verwendet XML-Verschlüsselung zur Verschlüsselung des SOAP-Envelope, XML-Signatur zur Gewährleistung der Integrität und den Security Token Service (STS) [15] als Public Key Infrastructure (PKI) mit X.509-Zertifikaten oder Kerberos-Token etc. WS-Trust [16] wird für die Ausgabe, Validierung und den Widerruf dieser Sicherheitstoken verwendet. Weitere Spezifikationen wie WS-SecureConversation [16] bieten einen Sicherheitskontext durch Austausch von Verschlüsselungsschlüsseln und Reduzierung des Overheads oder WS-SecurityPolicy [17] zur Definition und Erlangung der Sicherheitsrichtlinien eines WS.

REST kann nur durch die Verwendung von TLS Punkt-zu-Punkt-Sicherheit bieten. Innerhalb der kommunizierenden Anwendung müssen zusätzliche Sicherheitsmerkmale realisiert werden.

11.9 Physikalische Anforderungen

Die Extraktion der SOAP-Envelope aus dem SOAP-Paket und das anschließende Parsen der XML-Informationen ist sehr zeitaufwendig. Darüber hinaus ist SOAP an XML gebunden und die XML-Daten selbst sind oft unnötig komplex und lassen sich in ungeeigneten Szenarien nicht optimieren [18]. SOAP ist daher auf eingeschränkten Geräten nur schwer zu realisieren.

Da **REST** mit verschiedenen, weniger umfangreichen Messaging-Formaten wie JSON verwendet werden kann, kann es zusätzliche Rechenleistung und Bandbreitenauslastung sparen. XML ist eine Auszeichnungssprache und wurde optimiert, um zusätzliche Informationen zu frei fließendem Text hinzuzufügen, wie z.B. Formatierung. JSON ist für die Darstellung von Objekten konzipiert. Eine ist für bestimmte Aufgaben besser geeignet als die andere. Je nach Anwendungsfall kann **REST** daher eine höhere Flexibilität bieten. Aufgrund seiner Zustandslosigkeit ist der Server nicht verpflichtet, den Kontext des Clients zwischen den Anfragen zu speichern und weitere Ressourcen zu sparen. Im Allgemeinen ist REST im Vergleich zu SOAP für eingeschränkte Geräte besser geeignet, aber es basiert immer noch auf HTTP, was für viele IoT-Geräte immer noch ein Problem darstellt.

11.10 Verfügbare Implementierungen

SOAP ist ein offener Standard und ist in verschiedenen Frameworks für verschiedene Programmiersprachen erhältlich. Z.B. .NET Framework für C# und VB.NET, Apache Axis für Java und C++, WSO2 für PHP etc.

Da **REST** ein allgemeines Programmierparadigma ist, hängt es nicht von einer Programmiersprache ab. Infolgedessen gibt es verschiedene Open-Source-Frameworks zur Erstellung von RESTful-APIs wie

hapi.js [5] für JavaScript, Jersey [6] für Java, Apigility [7] für PHP, Grape [8] für Ruby, asp.net [9] für .NET etc.

11.11 Referenzen

[1] World Wide Web Consortium (W3C), "Web Services Activity". [Online]. Verfügbar unter: https://www.w3.org/2002/ws/. [Zugriff: 23-Aug-2017].

Microsoft, "Web Services Specifications Index Page". [Online]. Verfügbar unter: https://msdn.microsoft.com/en-us/library/ms951274.aspx. [Zugriff: 23-Aug-2017].

[3] OASIS, "Normen." [Online]. Verfügbar unter: https://www.oasis-open.org/standards#oasiscommiteespecs. [Zugriff: 23-Aug-2017].

[4] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", University of California, Irvine, 2000.

[5] "hapi - Ein reichhaltiges Framework für die Erstellung von Anwendungen und Diensten." [Online]. Verfügbar unter: https://hapijs.com/. [Zugriff: 16-Nov-2017].

[6] "Jersey - RESTful Web Services in Java." [Online]. Verfügbar: https://jersey.github.io/. [Zugriff: 16-Nov-2017].

[7] Zend Technologies, "Fähigkeit - Dokumentation". [Online]. Verfügbar unter: https://apigility.org/documentation. [Zugriff: 23-Nov-2017].

"Rubine Traube", Dez-2017. [Online]. Verfügbar unter: http://www.ruby-grape.org/. [Zugriff: 23-Nov-2017].

Microsoft, "ASP.NET Übersicht". [Online]. Verfügbar unter: https://docs.microsoft.com/en-us/aspnet/overview. [Zugriff: 23-Nov-2017].

[10] W. Colitti, K. Steenhaut, N. De Caro, B. Buta und V. Dobrota, "Evaluation of constrained application protocol for wireless sensor networks", *IEEE Work. Lokaler Metrop. Bereichsnetze*, Nr. 7, 2011.

[11] R. Bilorusets *et al.*, "Web Services zuverlässiges Messaging-Protokoll (WS-ReliableMessaging)", *Specif. BEA, IBM, Microsoft TIBCO*, Nr. Februar 2005.

[12] World Wide Web Consortium (W3C), "Neueste SOAP-Versionen". [Online]. Verfügbar unter: https://www.w3.org/TR/soap/. [Zugriff: 25-Aug-2017].

[13] Computer Projects of Illinois Inc., "Puting SOAP to REST", 2015.

[14] K. Lawrence, C. Kaler, A. Nadalin, R. Monzillo und P. Hallam-baker, "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)," *OASIS Stand. Spezifisch.* Nein, nein. Februar 2006.

Microsoft Corporation, "Brokered Authentication: Security Token Service (STS)," Dez-2005. [Online]. Verfügbar unter: https://msdn.microsoft.com/en-us/library/ff650503.aspx. [Zugriff: 28-Aug-2017].

[16] K. Lawrence *et al.*, "WS-SecureConversation 1.3", *OASIS Stand. Spezifisch.* Nein, nein. März 2007.

[17] K. Lawrence *et al.*, "WS-SecurityPolicy 1.2", *OASIS Stand. Spezifisch.* Nein, nein. Juli 2007.

[18] A. Mani und A. Nagarajan, "Understanding quality of service for web services", 01-Jan-2002. [Online]. Verfügbar unter: https://www.ibm.com/developerworks/library/ws-quality/index.html. [Zugriff: 25-Aug-2017].

[19] T. Nixon, A. Regnier, D. Driscoll und A. Mensch, "Geräteprofil für Web Services Version 1.1", *OASIS Stand. Spezifisch.* Nein, nein. Juli 2009.