Whitepaper

Datenaustausch im Zeitalter des „Industrial Internet of Things“

Markus Weishaar | Produktmanager IIoT
Dunkermotoren GmbH

dunkermotoren | AMETEK® ADVANCED MOTION SOLUTIONS

Durch die Etablierung des “Industrial Internet of Things” (IIoT) als Schlüsseltechnologie der vierten industriellen Revolution rücken von Maschinen und Komponenten erzeugte Daten immer stärker in den Fokus von Unternehmen. Die Daten werden nicht von ungefähr als neuer Rohstoff und auch als Gold des digitalen Zeitalters bezeichnet. Denn durch die Analyse der Daten lassen sich Informationen zu den zugehörigen Produkten und Prozessen gewinnen, was wiederum die Grundlage für Produktivitätssteigerungen oder auch komplett neue Geschäftsmodelle ist. Werden zum Beispiel die Betriebsdaten von Motoren analysiert, lassen sich daraus auf der einen Seite dienstleistungsbezogene Geschäftsmodelle wie „Predictive Maintenance“ ableiten, was auf der anderen Seite Produktionsausfälle minimiert und somit die Produktivität erhöht. Doch wie finden die Daten ihren Weg vom Gerät, das Daten erzeugt, zum Ort der Analyse? Dieser Frage widmet sich dieses White-Paper und beleuchtet die gängigsten Kommunikationsstandards im Kontext des IIoT.

Unbestritten ist inzwischen, dass klassische Feldbus-Systeme wie DeviceNet oder Profibus durch industrielle Ethernet Busse wie Profinet, EtherCAT oder Ethernet/IP mittelfristig abgelöst werden. Dies belegt auch die jährliche Betrachtung der Marktanteile von Bussystemen durch die Firma HMS, in der 2019 klassische Feldbusse erstmals rückläufig sind (Abbildung 1). Somit ist zumindest klar, dass analog zur IT auch in der Automatisierung Ethernet als Übertragungsmedium bei kabelgebundenen Verbindungen zukünftig gesetzt ist. Dennoch beherrschen mit den „Industrial Ethernets“ weiterhin proprietäre Standards die Feld- und Steuerungsebene. Das erschwert aktuell noch deutlich eine herstellerübergreifende Kommunikation. Doch auch hier zeigt sich durch breite Unterstützung von OPC UA als einheitlichen Standard Licht am Ende des Tunnels, auf dem Weg hin zu einer gemeinsamen und offenen Kommunikation in horizontaler und vertikaler Richtung. Das ist entscheidend, da eine einheitliche und offene Kommunikation zwischen Geräten und Systemen sowohl die größte Herausforderung als auch der größte Enabler für das „Industrial Internet of Things“ ist. Denn nur durch eine hersteller- und ebenenunabhängige Kommunikation haben Maschinen- und Anlagenbauer die Möglichkeit, die für Ihren Anwendungsfall besten Technologien auszuwählen und zu kombinieren. Und nur so ist es möglich, Daten wirtschaftlich sinnvoll von allen Feldgeräten in übergeordnete Analysesoftware zu übermitteln, da diese nicht aus den unterschiedlichsten proprietären Systemen zusammengetragen und übersetzt werden müssen.
Neben diesen Kommunikationsstandards, die ihren Ursprung in der Automatisierung haben, gewinnen auch die aus der IT stammenden Ansätze des Datenaustauschs via API (application program interface) oder via Protokollen wie MQTT und AMQP immer mehr Bedeutung in der Automatisierung, da es immer stärker darum geht, Daten von Automatisierungsgeräten in IT-Systeme zu übermitteln.

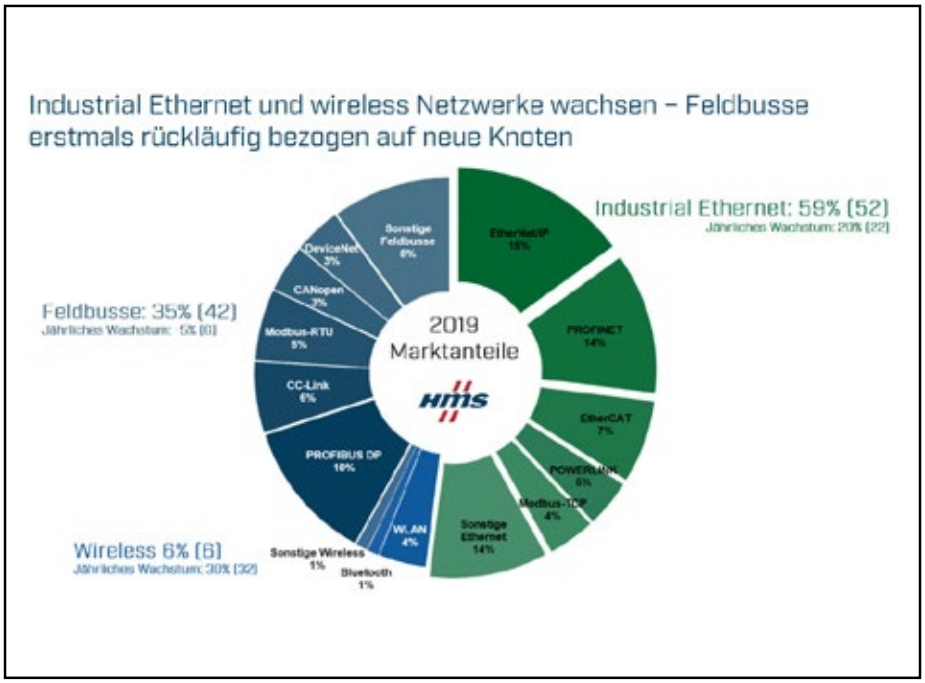

Abbildung 1: Marktanteile industrielle Netzwerke[1]

Autor: Markus Weishaar
Datum: 18.02.2020

Nachfolgend wird detailliert auf die einzelnen Technologien eingegangen und deren Vor- und Nachteile aufgezeigt:

Industrial Ethernet:

Bei den „Industrial Ethernets“ handelt es sich um herstellergetriebene proprietäre Kommunikationsprotokolle, die auf dem Ethernet-Standard IEEE 802.3 aufsetzen und diesen um Echtzeitfähigkeit und verschiedene weitere Steuerungsfunktionalitäten wie zum Beispiel “Motion” erweitern. Die drei am weitestenverbreiteten Protokolle sind EtherNet/IP (Rockwell), Profinet (Siemens) und EtherCAT (Beckhoff). Wie stark die einzelnen „Industrial Ethernets“ hier auf das herkömmliche Ethernet zurückgreifen variiert von System zu System. Die ersten beiden nach IEEE 802.3 standardisierten Schichten des OSI Modells, also die physische Übertragungsschicht und die MAC-Adressierung, werden praktisch von allen Protokollen verwendet. Inwieweit aber darauf aufbauend auf die Standardprotokolle IP, TCP und UDP des herkömmlichen Ethernets zurückgegriffen wird hängt stark vom jeweiligen „industrial Ethernet“ ab. EtherNet/IP ist dabei am nächsten am herkömmlichen Ethernet , da hier auch die Standardprotokolle IP, TCP und UDP fixer Bestandteil des Protokollstacks sind und erst oberhalb der Transportschicht das EtherNet/IP spezifische CIP-Protokoll aufsetzt. Profinet stellt hier schon oberhalb der MAC-Adressierung einen eigenen Protokollstack dar, bietet beim Verbindungsaufbau und bei azyklischen Diensten aber noch die Möglichkeit, auf UDP/IP zurückzugreifen. EtherCAT hingegen ist oberhalb von Schicht 2 ein komplett eigenständiges Protokoll und nutzt lediglich den Ethernet-Frame. Dies ist zum einen der Grund für die herausragende Performance von EtherCAT in Bezug auf die Latenzzeiten für das es entwickelt wurde, aber auch dafür, dass EtherCAT am weitesten entfernt ist vom herkömmlichen Ethernet. Die Protokolle sind zwar prinzipiell offen und können auch von Drittanbietern in Ihre Geräte implementiert werden. Ihre volle Performance entfalten die Protokolle aber meist nur im Ökosystem der Hersteller, die die Protokolle entwickelt haben. Dies macht sich dadurch bemerkbar, dass oft Lizenzen erforderlich sind und speziell bei der Integration von Masterfähigkeit gewisse Hürden existieren. Die „Offenheit“ dient somit eher dazu, Geräte von Drittanbietern in das eigene Ökosystem zu integrieren und Anwender im eigenen Ökosystem zu binden. Aufgrund des Entwicklungszyklus von ~15 Jahren innerhalb der Automatisierung spielen die „Industrial Ethernets“ auch die nächsten Jahre noch eine wichtige Rolle, selbst wenn sich z. B. mit OPC UA ein übergreifendes System etablieren würde. Die Annahme wird dadurch gestützt, dass entsprechend der HMS Studie die „Industrial Ethernets“ erst 2018, also ebenfalls ~ 15 Jahre nach der ersten Einführung, die klassischen Feldbusse bei der Anzahl neuer Knoten überholt haben. Zudem waren die klassischen Feldbusse auch erstmals 2019 rückläufig.

OPC UA:

OPC UA ist die Abkürzung für **O**pen **P**latform **C**ommunication **U**nified **A**rchitecture und steht, wie der Name schon sagt, für eine offene Kommunikation mit vereinheitlichtem Aufbau. OPC UA ist dabei nicht einfach als zusätzliches Kommunikationsprotokoll zu sehen, sondern viel mehr als Kommunikations-Framework, dass neben der Datenübertragung auch die Bedeutung von und den Zugriff auf Daten beschreibt und gleichzeitig Sicherheitsmechanismen mitbringt. Die Sicherheit wird hierbei über Zertifikate und eine Zertifikats- und Zugriffsrechtsverwaltung realisiert, welche OPC UA mitbringt. Hier muss allerdings beachtet werden, dass OPC UA nicht per se sicher ist, sondern die nötigen Mechanismen und Eigenschaften mitbringt , auf deren Basis eine sichere Kommunikation implementiert werden kann. OPC UA basiert darauf, dass Geräte in Form von Informationsmodellen abgebildet werden. Das bedeutet, das Geräte in Objekten mit Ihren zugehörigen Variablen, Methoden, Events und Ihrer Beziehung zu anderen Objekten beschrieben werden. Ein Motor kann beispielswiese in der einfachsten Form ein einzelnes Objekt mit Variablen, Events und Methoden sein, sich aber auch aus verschiedenen Objekten wie z. B. Strom-, Positions- und

Autor: Markus Weishaar
Datum: 18.02.2020

Geschwindigkeitsregler zusammensetzen. Somit können beliebig komplexe Komponenten und Maschinen semantisch abgebildet werden und es ist möglich, dass eine OPC UA Anwendung diese semantischen Modelle versteht, ohne sie im Vorfeld zu kennen. Damit dies herstellerübergreifend funktioniert, setzt OPC UA auf eine service-orientierte Architektur (SOA), die den Zugriff auf Informationsmodelle über standardisierte Services definiert. In der Basisimplementierung bringt OPC UA die Informationsmodelle Data Access (DA), Alarms & Conditions (AC), Historical Access (HA) und Programs, sowie die Services Browse, Read/Write, Methodenaufruf und Subscribe auf einzelne Variablen mit. Darüber hinaus kann die Funktionalität von OPC UA über branchenspezifische Informationsmodelle, die sogenannten „Companion Specifications“ erweitert werden. Um auch innerhalb von Branchen wie z. B. der Antriebstechnik noch den unterschiedlichen Differenzierungsmerkmalen der einzelnen Hersteller Rechnung zu tragen, gibt es die Möglichkeit zusätzliche, nicht standardisierte Funktionen über die herstellerspezifischen Erweiterungen in OPC UA abzubilden (Abbildung 2).

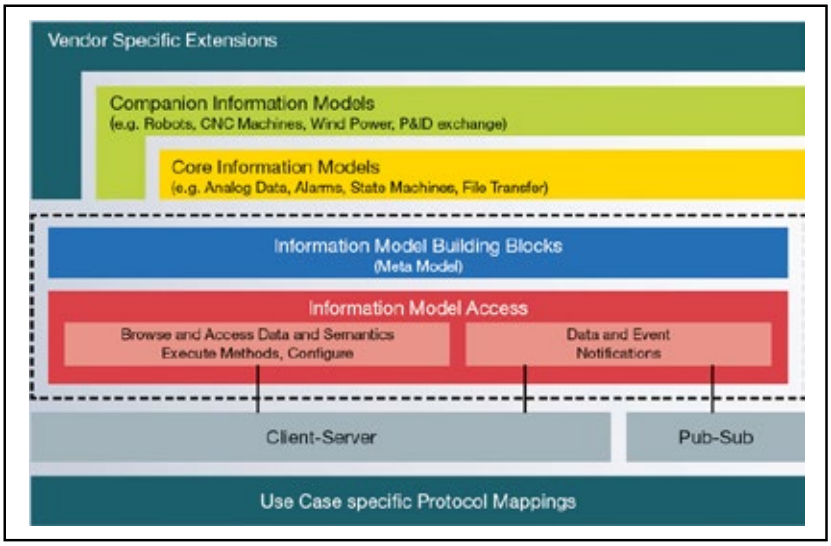

Abbildung 2: OPC UA Aufbau[2]

Für die Kommunikation selbst stellt OPC UA zwei Arten zur Verfügung. Server/ Client und Publish/ Subscribe. Dabei können beide Kommunikationsarten parallel in einer OPC UA Anwendung verwendet werden und auch jede Anwendung jede Rolle einnehmen (Abbildung 3). Somit deckt OPC UA sowohl in der Automatisierung übliche direkte Verbindungen über Server/ Client, als auch in der Cloud-Anbindung übliche indirekte Verbindungen über Publish/ Subscribe ab. Bei Publish/ Subscribe setzt OPC UA dabei auf die Einbettung von hier gängigen Standards wie MQTT oder AMQP. Im Kontext Publish/ Subscribe auf der Feldebene wird auch eine Variante mit UDP als Übertragungsprotokoll unterstützt.

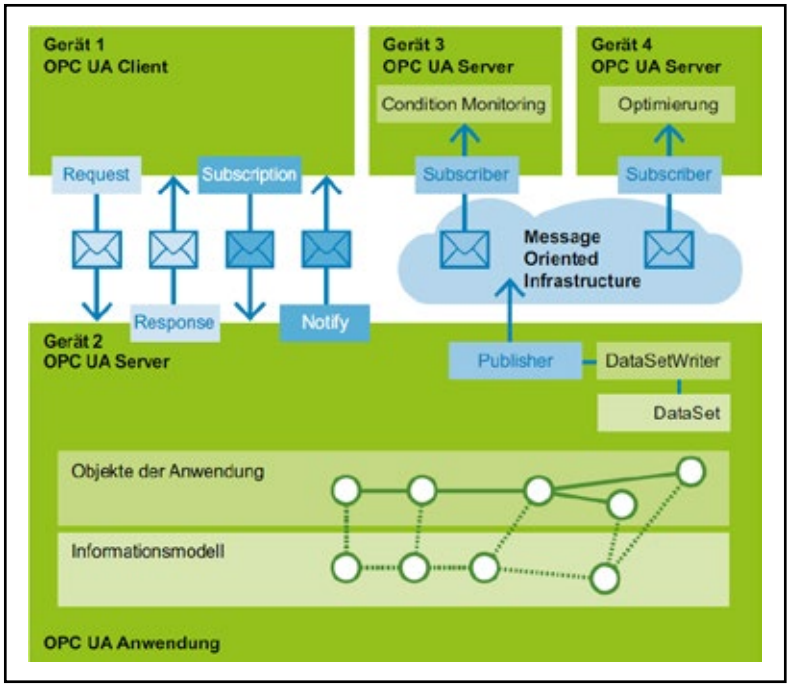

Abbildung 3: OPC UA Kommunikation[3]

Auf der Transportschicht der Kommunikation basiert OPC UA immer auf dem Internet Protocol (IP) und benötigt daher immer eine Ethernet-basierte Netzwerkinfrastruktur. Daher ist OPC UA in seiner aktuellen Form eine ideale Ergänzung für „Industrial Ethernets“, um Daten parallel zur Steuerungskommunikation in übergeordnete Systeme für Analysen und Monitoring zu

Autor: Markus Weishaar
Datum: 18.02.2020

übertragen. Aktuell wird daran gearbeitet, OPC UA um das Time-Sensitive Networking (TSN) zu erweitern und somit eine deterministische Echtzeitkommunikation zu ermöglichen. Dieser Ansatz, sowie auch 5G, sind auch die Basis für die „Field-Level-Communication“ (FLC) Initiative der OPC-Foundation, die daran arbeitet, die OPC UA Kommunikation auf die Feldebene zu bringen. OPC UA als Ethernet-basierter und deterministischer Kommunikationsstandard auf der Feldebene hat dann das Potential, eine einheitliche und herstellerübergreifende Kommunikation zu ermöglichen. Gerade für dezentrale Systemarchitekturen ohne ausgeprägte Steuerungsebene ist das ein vielversprechender Technologieschritt, da Feldgeräte so auf die gleiche Art und Weise untereinander und zu Leit- und Managementsystemen kommunizieren können. Wie beschrieben, eignet sich OPC UA aufgrund seiner Features als ideales Toolkit für eine hersteller- und ebenenübergreifende Kommunikation. Dennoch gibt es auch hier einige Einschränkungen zu beachten. Der hohe Abstraktionsgrad von OPC UA der nötig ist, um den generischen Ansatz zu ermöglichen, erschwert natürlich den Einstieg und auch die Analyse der Verbindung selbst. Zudem führen die ganzen Features, die OPC UA mitbringt auch zu einer entsprechenden Anforderung an die Hardware, auf der eine OPC UA Anwendung laufen soll. Aus diesem Grund gibt es auch reduzierte OPC UA Profile, die keine Sicherheitsmechanismen unterstützen oder in der untersten Stufe nur eine Verbindung zulassen ohne Methodenaufrufe und Subscriptions.

MQTT:

Message **Q**ueuing **T**elemetry **Tr**ansport ist ein leichtgewichtiges Protokoll zur Datenübertragung. Im besten Fall können Pakete von lediglich 2 Byte realisiert werden, was speziell bei einer Vielzahl von Geräten und Nachrichten von Vorteil ist. Das, zusammen mit der einfachen Implementierung von MQTT, hat in den vergangenen Jahren dazu geführt, dass MQTT im IoT Umfeld sehr starke Verwendung gefunden hat. Allerdings ist bei MQTT zu beachten, dass es entgegen OPC UA ein reines Übertragungsprotokoll ist und kein erweitertes Framework liefert mit Funktionen wie einer semantische Datenbeschreibung oder Sicherheitsmechanismen. Das bedeutet, dass Sicherheitsmechanismen zur Absicherung der Verbindung separat implementiert werden müssen und auf beiden Seiten der Kommunikation deklariert werden muss, um was für eine Art von Daten es sich handelt und wie diese zu verstehen sind. Vom Aufbau her ist MQTT ein offenes Publish/ Subscribe Protokoll für indirekte 1 zu n Kommunikation. Das bedeutet, dass ein Publisher basierend auf Events Nachrichten mit einem bestimmten Topic an einen sogenannten Broker sendet. Der Broker leitet die entsprechende Nachricht an alle Subscriber weiter, die das entsprechende Topic abonniert haben. Bezogen auf einen Motor kann das z. B. bedeuten, dass dieser unter dem Topic „Diagnose/Überstrom “ seine Serialnummer und den zugehörigen Wert an einen Broker übermittelt, wenn der eingestellte Grenzwert überschritten wird. Der Broker leitet die Nachricht dann an alle Abonnenten wie beispielsweise mobile Endgeräte von Servicetechnikern, Leitsysteme oder Cloud-Applikationen weiter. Bei der Konfiguration der Publish/ Subscribe Kommunikation bietet MQTT noch einige nützliche Features. Retained Messages ermöglichen es zum Beispiel, dass die letzte gesendete Nachricht zu einem Topic beim Broker hinterlegt bleibt und einem neuen Subscriber direkt bei Anmeldung übermittelt wird. Daneben gibt es noch eine Last Will Nachricht, die ein Publisher beim Broker hinterlegen kann. Diese wird an alle Subscriber verschickt, wenn ein Gerät nicht mehr verbunden ist und sich zuvor nicht richtig abgemeldet hat. Auch wenn auf Seiten der Subscriber Verbindungsabbrüche vorkommen bietet, das Feature Persistent Session die Möglichkeit, dass verpasste Nachrichten im Broker gepuffert werden und dem Subscriber bei erneuter Anmeldung übermittelt werden. Als letztes bieten verschieden Quality of Service Einstellungen noch drei Möglichkeiten, um sicher zu gehen, dass versendete Nachrichten bei genau einem, bei mehr als einem oder bei mindestens einem Subscriber angekommen sind. MQTT ist somit ein leicht zu beherrschendes Protokoll, dass sich ideal eignet, auf sehr ressourcenarmen Geräten implementiert zu werden und auch bei Unterbrechungen der Verbindung gewährleisten kann, dass die Daten Ihr

Ziel erreichen. Allerdings muss auf beiden Seiten bekannt sein, welche Daten kommuniziert werden und für die Datensicherheit vor allem des Brokers muss auf anderen Wegen gesorgt werden.

AMQP:

Das **A**dvanced **M**essage **Q**ueuing **P**rotocol ist neben MQTT das am weitesten verbreitetste Kommunikationsprotokoll im IoT-Umfeld. AMQP arbeitet dabei wie MQTT ebenfalls mit einem Broker und dem Publish/ Subscribe Prinzip. Bei AMQP besitzt jeder Subscriber eine Warteschlange, in die Nachrichten mit abonnierten Topics vom Broker abgelegt werden. Die Nachrichten bleiben so lange in einer Warteschlange bis der Subscriber bestätigt hat, dass er die Nachricht empfangen hat. Die Warteschlangen sind somit auch ein Puffer für Nachrichten, falls ein Subscriber nicht immer verbunden ist. Kann eine Nachricht nicht an einen Empfänger übermittelt werden, bekommt der Publisher eine entsprechende Nachricht. Neben Publish/ Subscribe bietet AMQP aber noch die weiteren Übertragungsarten:

» Fanout, bei dem der Broker eine Nachricht an alle verbundenen Warteschlangen übermittelt.

» Direct, bei der über einen Identifier eine feste Verbindung zwischen einem Subscriber und einer Warteschlage hergestellt werden kann.

» Headers, bei der die Verteilung von Nachrichten im Broker über Nachrichtenheader statt Identifier erfolgt und das gegenüber Direct mehr Möglichkeiten bei der Regelerstellung bietet.

Zudem können bei AMQP die Nachrichten auch um Meta-Daten ergänzt werden, die die Daten der Nachricht in Form von Attributen beschreiben und die vom Empfänger genutzt werden können.
Der größte Unterschied zu MQTT stellt somit der erweiterte Funktionsumfang bei der Nachrichtenübermittlung von AMQP dar. Dieser bringt aber auch einen höheren Implementierungsaufwand und einen größeren Ressourcenbedarf mit sich. Die kleinste mögliche Paketgröße bei AMQP beträgt bereits 60 Byte. Daher gilt hier die Abwägung, ob die erweiterte Funktionalität von AMQP benötigt wird oder doch das noch einfachere MQTT ausreicht.

API/ REST API:

Application **P**rogram **I**nterfaces entstammen dem Ansatz, Programme in funktionsbasierte Module zu unterteilen. Die einzelnen Module stellen Ihre öffentlichen Daten anderen Module über APIs zur Verfügung und holen benötigte Daten bei APIs anderer Module ab. APIs sind Strukturen mit verschiedenen Variablen, die vom zugehörigen Modul beschrieben werden und von anderen Modulen gelesen werden können. Sie entkoppeln somit den „privaten“ Code der Module von der Außenwelt. Das ermöglicht es, leichter wartbare Module zu erzeugen und fehlerhaften Code schneller zu identifizieren, da jedes Modul durch das Beschreiben der API mit den geplanten Kommandos und das Prüfen der erwarteten Ergebnisse für sich getestet werden kann. APIs lassen sich generell somit auf jeden Anwendungsfall exakt zuschneiden, sind aber hersteller-, anwendungs- und modulspezifisch und nicht standardisiert. Daher müssen APIs beschrieben werden, wie sie erreicht werden, welche Variablen sie enthalten und ob die Variablen nur Lese- oder auch Schreibrechte haben. Gerade beim Datenaustausch mit anderen Herstellern über APIs oder bei öffentlichen APIs ist die detaillierte Beschreibung wichtig, damit die „externen“ Programmierer wissen, wie sie die Schnittstelle nutzen können, da Ihnen das Wissen fehlt wie die hinter der API liegende Applikation arbeitet.

steht für **Re**presentational **S**tate **T**ransfer. REST ist kein eigener Standard oder ein Protokoll, sondern ein Architekturansatz für die Kommunikation innerhalb von verteilten Systemen. Da REST kein ausspezifizierter Standard ist, gibt es somit nicht im Detail vor, wie konforme Implementierungen aussehen müssen, sondern gibt sechs Architekturprinzipien („Constraints“) vor, die eingehalten werden müssen. Technologisch setzt REST auch auf Bestehendes, so kommt bei der Übertragung oft HTTP/S als Protokoll und XML (Extensible Markup Language) oder JSON (Java Script Object Notation) als Datenformat für Informationen zum Einsatz. Da REST im Jahr 2000 entwickelt wurde während des großen Durchbruchs des Internets, liefert das selbige bereits einen großen Teil der für REST nötigen Infrastruktur und die meisten Webservices basieren auf REST. Die sechs von REST definierten Architekturprinzipien sind dabei:

REST baut auf ein **Client-Server-Modell** mit strikter Trennung von Datenhaltung und User-Interface. Das bedeutet, dass User-Interfaces als Clients leicht an unterschiedliche individuelle Rahmenbedingungen angepasst werden können während die Datenhaltung als Server durch einen standardisierten Aufbau leicht zu skalieren ist.

Nachrichten müssen **zustandslos („stateless“)** sein. Eine Anfrage des Clients beim Server muss somit in sich geschlossen sein und alle Information zum Applikationszustand beinhalten. Der Kontext der Nachricht muss also immer mitgeliefert werden, da es bei REST keine bestehenden Sitzungen gibt und der Server diese ansonsten nicht interpretieren kann. Dieses Prinzip sorgt ebenfalls für eine einfachere Skalierbarkeit, da verschiedene Nachrichten des Clients von unterschiedlichen Servern bearbeitet werden können.

Der Client hat die Möglichkeit, Antworten des Servers für eine erneute identische Anfrage zu **Puffern („Cachen“)** sofern diese entsprechend gekennzeichnet sind. Dies dient dazu, den Traffic auf dem Netzwerk zu verringern und die Effizienz des Netzwerks zu erhöhen. Allerdings besteht das Risiko, dass der Client so auf veraltete Daten zurückgreift.

REST setzt auf eine **einheitliche Schnittstelle** zwischen allen Clients und Servern mit einheitlichen Protokollen, Datenformaten und Methoden zum Zugriff. Mit der Verwendung von einheitlichen Schnittstellen gehen in der Regel Performanceeinbußen einher, da alle Daten in ein einheitliches Format gewandelt werden muss. Diese Einbußen nimmt man für eine einfachere Architektur und Usability aber gerne in Kauf. Ein beispielhafter Aufbau wäre die Verwendung von HTTP/S zur Übertragung und JSON als Datenformat sowie der folgenden gängigen Methoden:

» GET - fordert Daten vom Server an
» POST - übermittelt Daten an den Server
» PUT/ PATCH - ändern bestehende Daten auf dem Server
» DELETE - löscht bestehende Daten auf dem Server

REST gibt eine Architektur in einem **Schichtensystem** vor mit klarer hierarchischer Struktur und Abgrenzung zwischen den Schichten. Dieser Ansatz ermöglicht es, stärker zu abstrahieren und so dem Anwender über eine einheitliche Schnittstellenschicht Zugriff auf unterschiedliche dahinterliegende Architekturen zu geben ohne dass er diese kennen muss. Dadurch ist es z. B. möglich, Legacy Systeme als Schicht zu Kapseln und über „neue“ Schnittstellen erreichbar zu machen. Dadurch ergibt sich eine erhöhte Sicherheit und Usability, allerdings aber auch durch die Abstrahierung ein größerer Overhead und größere Latenzzeiten durch die Kommunikation über mehrere Schichten.

Autor: Markus Weishaar
Datum: 18.02.2020

Als einziges optionales Prinzip bei REST bietet **„Code on Demand“** die Möglichkeit über die API ausführbaren Code an den Client zu übermitteln bzw. nachzuladen. Dies gibt die Möglichkeit, die Funktion von einem Client unabhängig von seinem eigenen Code zu verändern bzw. zu erweitern.

Fazit:
Zusammenfassend kann gesagt werden, dass es auch in Zukunft nicht den einen Kommunikationsstandard über alle Ebenen geben wird, sich die Vielfalt aber stark reduzieren wird. Am ehesten hätte OPC UA das Potential eine vertikale und horizontale Kommunikation auf und über alle Ebenen zu ermöglichen. Allerdings stellt OPC UA für ein solches Szenario auch gewisse Anforderungen an die Hardware in Form von Speicher, Rechenleistung oder auch Kryptochips, die von aktuellen Embedded Geräten meist noch nicht erfüllt werden können. Auch bezüglich TSN ist aktuell noch nicht genau absehbar, wann und in welcher Form OPC UA auf der Feldebene als Echtzeitbus eingesetzt werden kann. Aktuell und über die nächsten Jahre werden auf der Feldebene noch die heterogenen „Industrial Ethernet“-Feldbussen dominieren, da hier auch der Investitionszyklus von ca. 15 Jahren bei Maschinen und Anlagen berücksichtigt werden muss, in dem bestehende Anlagen Schrittweise durch neue Anlagen ersetzt werden. Hier ist es eher wahrscheinlich, dass OPC UA als einheitliche Ethernet Schnittstelle zur EDGE zum Tragen kommt, um nicht jedes Protokoll für jedes Gerät anbinden zu müssen. Aber auch im Bereich der Cloud Kommunikation haben sich Publish/ Subscribe Protokolle wie MQTT oder AMQP Datenaustausch über REST APIs inzwischen als quasi Standard für etliche Applikationen etabliert, was eine kurzfristige Ablösung unwahrscheinlich macht.
Basierend auf dieser Situation und den gegebenen Randbedingungen scheint die nachfolgend beschriebene Kommunikationsarchitektur als wahrscheinlich für die Zukunft:

Daten von Feldgeräten werden unabhängig vom „Industrial Ethernet“ Feldbus über OPC UA oder alternativ auch direkt über den jeweiligen „Industrial Ethernet“ Feldbus an die EDGE übertragen. Dort werden die ankommenden Daten über Software-Adapter in die nötigen Internet-Protokolle wie MQTT oder AMQP umgesetzt oder direkt mittels einer REST API übergeben. Welcher Weg hier exakt zum Einsatz kommt hängt hier stark damit zusammen, wohin die Daten übermittelt werden bzw. in welchem IIoT-Ökosystem gearbeitet wird. Innerhalb des Cloud-Levels ist aktuell ein Datenaustauschs via REST API am wahrscheinlichsten.
Auf Seiten der Übertragung kommen somit auf den unterschiedlichen Schichten auch zukünftig unterschiedliche Technologien zum Einsatz, innerhalb der Ebenen gibt es aber eine stärkere Standardisierung. Beim Datenformat hingegen zeichnet sich eine stärkere Standardisierung über alle Ebenen hinweg ab. Hier hat aktuell die von OPC UA definierte semantische Datenbeschreibung in JSON das Potenzial der durchgehende quasi Standard zu werden.

Quellenangabe:
[1] https://www.hms-networks.com/de/news/pressemitteilungen-von-hms/2019/05/07/marktanteile-industrieller-netzwerke-2019-aus-sicht-von-hms
[2] OPC Unified Architecture, Interoperability for Industrie 4.0 and the Internet of Things; Version 10 INA; OPC Foundation, 2019; S. 24
[3] Industrie 4.0 – Kommunikation mit OPC UA; VDMA; Fraunhofer IOSB-INA, 2017, S. 13

Ihr Kontakt für Public Relations:
Janina Dietsche | janina.dietsche@ametek.com
Tel: +49 (0)7703/930-546