Edge und Fog Computing Architekturen im Industrial IoT

Dipl.-Ing. Günter Obiltschnig
Applied Informatics Software Engineering GmbH
Maria Elend 143
9182 Maria Elend
Austria
guenter.obiltschnig@appinf.com

Einleitung

Edge und Fog Computing Architekturen ermöglichen es, die „Intelligenz“ von verteilten Systemen aus der Cloud heraus wieder näher zu den eigentlichen „Dingen“ zu bringen, also konkret in ein vernetztes Gerät selbst (Edge) oder in verteilte Netzwerkinfrastruktur (Fog). Im Folgenden wird zunächst die grundlegende Funktionsweise von Edge und Fog Computing erklärt. In weiterer Folge wird gezeigt, wie Edge und Fog Computing Konzepte im Industrial IoT Umfeld mit Hilfe von Open Source Technologien umgesetzt werden können. Dabei wird auch auf die Rolle unterschiedlicher Technologien wie OPC-UA, REST, Microservices, MQTT und DDS, sowie Security Aspekte eingegangen.

Ab in die Cloud?

Betrachtet man die Systemarchitektur verschiedener am Markt verfügbarer IoT Lösungen so findet man als Gemeinsamkeit häufig den Ansatz, Sensordaten so schnell und einfach wie möglich an ein IT System in der Cloud zu senden. Dort werden diese dann archiviert und mit „big data“ und „machine learning“ Technologien weiterverarbeitet, um daraus Erkenntnisse und Entscheidungen abzuleiten zu können.

Dies mag für viele, vor allem einfache Anwendungen funktionieren. Allerdings trifft man recht schnell auf die Grenzen dieses Ansatzes, im Hinblick auf Latenz, Netzwerkbandbreite, Verfügbarkeit, Zuverlässigkeit, Sicherheit und Skalierbarkeit. Werden lediglich alle paar Minuten z. B. Temperaturwerte oder andere einfache Messwerte gesendet um diese zu archivieren und auszuwerten mag das ganz gut funktionieren. Ganz anders sieht die Sache aus, wenn z. b. Vibrationsmessungen mit Hilfe von Beschleunigungssensoren gemacht werden. Hier kommen recht schnell größere Datenmengen zusammen (mehrere tausend Samples pro Sekunde), welche die Infrastruktur an ihre Grenzen bringen können. Eine lokale Verarbeitung solcher Daten ist unbedingt notwendig um die Belastung für Netzwerk und Backend zu reduzieren.

Edge Computing

Als Edge Computing bezeichnet man die dezentrale Datenverarbeitung am Rand des Netzwerks, der sogenannten Edge (engl. für Rand oder Kante). Edge Computing ist

kein wirklich neues Konzept, sondern vielmehr ein aus dem Produktmarketing kommender Begriff für einen bestimmten Systemarchitekturansatz, der durchaus schon lange in Verwendung ist. Im Wesentlichen geht es darum, einen Teil der Datenverarbeitung aus der Cloud heraus zu nehmen und lokal, auf einem System nahe am Ort des Geschehens, durchzuführen. Abbildung 1 zeigt einen Ansatz für Edge Computing im Industrial IoT Bereich. Edge Computing kann hier auf leistungsfähigen PLCs (SPS), PAC (Programmable Automation Controller) oder IPC (Industrie PC) Systemen zum Einsatz kommen. Eine wichtige Voraussetzung für Edge Computing ist die möglichst freie Programmierbarkeit dieser Systeme. Weiters muss es einfach möglich sein, die Programme zu modifizieren, bzw. zu aktualisieren, um rasch auf neue Anforderungen reagieren zu können. Dabei können z. B. *DevOps* Praktiken zum Einsatz kommen.

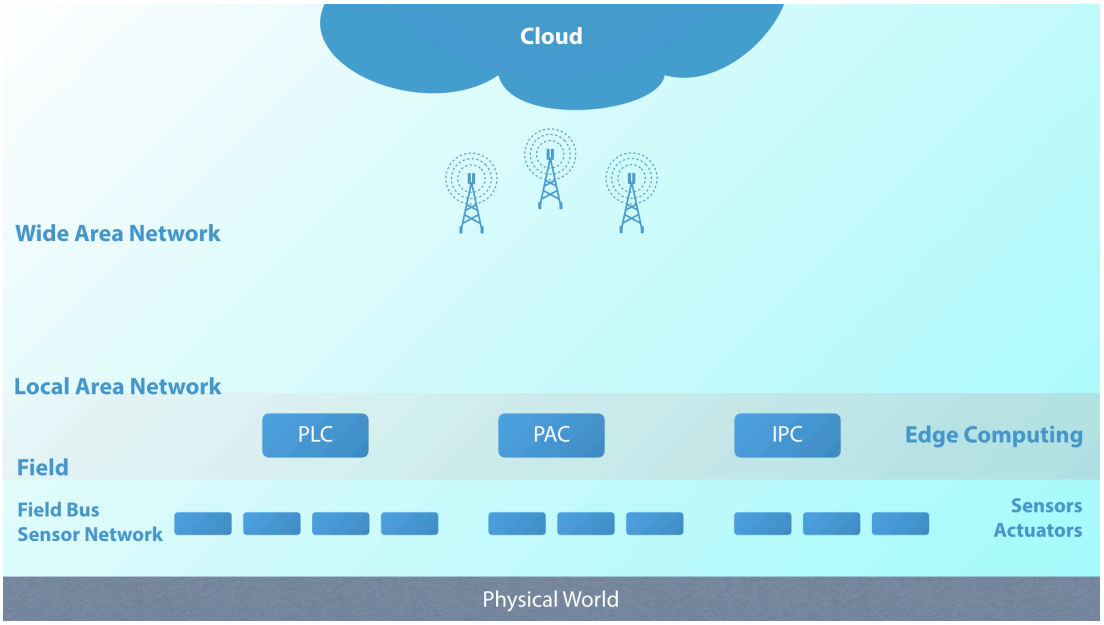

Abb. 1: Edge Computing im Industrial IoT

Fog Computing

Denkt man Edge Computing weiter und führt zwischen Edge und Cloud weitere Verarbeitungsschichten ein, kommt man zum Fog Computing. Fog (Nebel) ist dabei eine Anspielung auf den Begriff Cloud (Wolke) unter dem Aspekt dass sich Nebel näher am Boden (also der physikalischen Welt) befindet als Wolken.

Nach dem *OpenFog Consortium* [1] ist Fog Computing eine horizontale, also branchenunabhängige, Architektur auf Systemebene, welche Datenverarbeitung, Speicherung, Steuerung und Vernetzung näher zum Benutzer entlang des Cloud-to-Thing Kontinuums verteilt. Im Gegensatz zum Edge Computing findet die Verarbeitung und Speicherung der Daten also auf mehreren, hierarchisch strukturierten Knoten (*Fog Nodes*) statt, die untereinander, sowie mit der Cloud, kommunizieren. Dabei kommen aus dem Cloud Computing und DevOps Bereich bekannte Methoden und Konzepte wie Container, Virtualisierung und Orchestrierung zum Einsatz. Als Fog Nodes können Embedded Systeme (z. B. in Kameras oder intelligente Sensoren) und Edge Devices, aber auch Smartphones, PCs und lokale Serversysteme, sowie spezialisierte Computersysteme (z. B. für hardware-beschleunigte KI Algorithmen) zum Einsatz kommen.

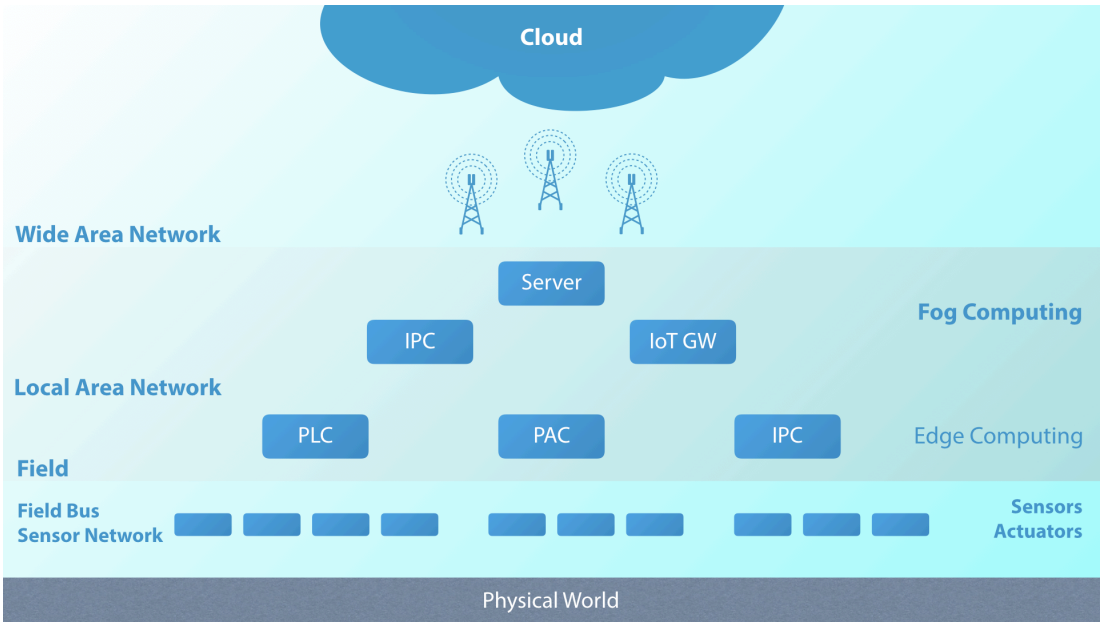

Abb. 2: Vom Edge zum Fog Computing

Vom OpenFog Consortium wurde eine Referenzarchitektur für Fog Computing geschaffen. Diese basiert auf acht Säulen:

- *Security* – betrachtet alle Aspekte der Datensicherheit und des Datenschutzes, wie Vertrauenswürdigkeit der Daten und beteiligten Systeme, sichere verschlüsselte Kommunikation, Zugriffskontrolle, sichere Softwareverteilung, usw.
- *Scalability* – befasst sich mit der Skalierbarkeit eines Systems im Hinblick auf möglichst lokale Datenverarbeitung und Steuerung und der Vermeidung unnötiger Netzwerkkommunikation. Es soll auch möglich sein, bei Bedarf die Leistungsfähigkeit eines Systems durch das Hinzufügen neuer Knoten zu steigern, ähnlich wie dies bei Cloud-Plattformen möglich ist.
- *Openness* – befasst sich mit Themen wie Interoperabilität, offenen Kommunikationsprotokollen, aber auch Ortstransparenz von Services.
- *Autonomy* – stellt sicher das System auch bei einem Ausfall einzelner Komponenten, z. B. der Netzwerkverbindung zur Cloud, oder eines Fog Nodes, bestmöglich weiterfunktioniert.
- *RAS* (*Reliability*, *Availability*, *Serviceability*) – spezifiziert Zuverlässigkeit, Verfügbarkeit und Wartbarkeit der Systeme.
- *Agility* – fordert, dass automatisierte Entscheidungen basierend auf einer Analyse von Daten jeweils möglichst nahe am Ort des Geschehens getroffen werden
- *Hierarchy* – befasst sich mit der möglichen Hierarchie der Systeme, bzw. Fog Nodes (z. B. Sensor, Fog Nodes auf mehreren Ebenen, Cloud)
- *Programmability* – stellt sicher das alle Fog Nodes programmierbar sind und an sich ändernde Anforderungen angepasst werden können.

Umsetzung

Edge und Fog Computing Architekturen können mit diversen etablierten und weitgehend auch als open source Implementierungen verfügbaren Technologien bereits heute umgesetzt werden.

REST und Microservices

HTTP und JSON-basierte Web Services, bzw. Programmierschnittstellen (APIs) nach dem REST Prinzip werden häufig zur Kommunikation zwischen Edge Device (IoT Gateway) und Cloud verwendet. Ebenso können diese auch zur Kommunikation von Fog Nodes untereinander verwendet werden. Die OpenAPI Spezifikation [2] ermöglicht eine maschinenlesbare, programmiersprachenunabhängige Beschreibung von REST APIs. Darauf aufbauend gibt es umfangreiche Werkzeugunterstützung z. B. zur automatischen Generierung von Client- und Server Code.

Auch Microservice-Architekturen (basierend auf REST, oder auch RPC Technologien wie *Apache Thrift*, *Protocol Buffers/gRPC* oder *ICE*) bieten sich für die Umsetzung von verteilten Diensten auf Fog Nodes geradezu an. Hierzu stehen eine Vielzahl an Frameworks und Libraries zur Unterstützung bei der Umsetzung zur Verfügung, die auch alle wichtigen Security-Aspekte (Authentifizierung, Autorisierung, Verschlüsselung) abdecken. Weniger geeignet sind diese Technologien zur reinen Verteilung von Daten. Hierfür sind Publish-Subscribe Technologien wie MQTT und DDS besser geeignet.

MQTT

MQ Telemetry Transport oder Message Queue Telemetry Transport) [3] ist ein offenes und standardisiertes Nachrichtenprotokoll für Machine-to-Machine-Kommunikation (M2M), das die Übertragung von Nachrichten zwischen Geräten, bzw. Systemen ermöglicht und auch bei unzuverlässigen Netzwerkverbindungen, hohen Latenzen und geringen Bandbreiten gut funktioniert. Ein Vorteil von MQTT ist, dass es nach dem Publish-Subscribe Schema funktioniert und so ein Sender die Empfänger seiner Nachrichten nicht kennen muss. Die Kommunikation läuft immer über einen Broker, der für die Weiterleitung der Nachrichten verantwortlich ist. Die Verteilung der Nachrichten erfolgt dabei über hierarchisch aufgebaute *Topics*. Jede Nachricht wird unter einem bestimmten Topic veröffentlicht. Clients, welche Nachrichten empfangen wollen, müssen sich für bestimmte Topics registrieren (*subscribe*), wobei hier auch Wildcards verwendet werden können. Die Struktur des Nachrichteninhalts (Payload) ist nicht definiert. In der Praxis wird häufig JSON verwendet, aber auch eine binäre Codierung der Daten ist möglich. Ein kleiner Nachteil von MQTT in Fog Computing Architekturen ist die Notwendigkeit für einen zentralen Broker. Die Kommunikation zwischen Clients und Broker kann verschlüsselt über TLS erfolgen. Authentifizierung und Autorisierung kann über Benutzername/Password, oder über X509 Zertifikate erfolgen, wobei die Verantwortung dafür beim Broker liegt. Unterschiedliche Implementierungen unterscheiden sich hier durch die Möglichkeiten (z. B. kann eingeschränkt werden, welche Clients sich auf welche Topics subscriben können). Ein großer Vorteil von MQTT ist die weite Verbreitung, die Verfügbarkeit mehrerer Implementierungen (sowohl Broker, als auch Clients für verschiedene Programmiersprachen und Plattformen), sowie die relativ einfache Nutzbarkeit.

DDS

Data Distribution Service [4] ist ein Middleware-Standard der Object Management Group, die auch z. B. die UML und CORBA Standards geschaffen hat. Wie MQTT liegt ein Publish-Subscribe Schema zu Grunde, welches sich allerdings wesentlich von MQTT unterscheidet. Nachrichten werden unter einem Topic veröffentlicht, wobei ein Topic aber auch die Struktur der Daten vorgibt. Topics und die Struktur der Daten werden über eine IDL-ähnliche Sprache beschrieben und sind somit relativ statisch. Clients können sich nicht nur für bestimmte Topics registrieren, sondern auch Filter definieren. Es wird praktisch ein in verschiedene „domains" unterteilter „global information space" gebildet, wo jeder Teilnehmer nur jene Daten erhält die er benötigt. Erzeuger und Verbraucher der Daten müssen sich dabei nicht kennen. Ein Vorteil von DDS ist, dass es ohne zentralen Broker funktioniert. Ein großer Nachteil ist die hohe Komplexität der Technologie, bedingt durch eine Vielzahl an Konfigurationsmöglichkeiten (speziell im Hinblick auf Security). Auch sind nur sehr wenige Implementierungen verfügbar, die meisten davon kommerziell. DDS eignet sich vorwiegend für die Kommunikation in lokalen Netzen, weniger für die Kommunikation mit Cloud Services – auch wenn es Bestrebungen gibt, dies zu ändern und entsprechende Erweiterungen des Standards zu schaffen. DDS unterstützt verschlüsselte Kommunikation, sowie X509 Zertifikate zur Authentisierung und Autorisierung (Zugriff auf Topics und Domains). Aufgrund der vollkommen verteilten Architektur ohne zentralen Broker ist dies allerdings sehr aufwändig in der Umsetzung.

OPC-UA

OPC-UA (OPC Unified Architecture) [5] ist ein Kommunikationsstandard (IEC 62541) aus dem Automatisierungsbereich, hervorgegangen aus der OPC Technologie, die auf der proprietären COM/DCOM Technologie von Microsoft basierte. Neben dem eigentlichen Kommunikationsprotokoll, das in zwei Varianten spezifiziert ist (TCP basiert mit binärer Codierung der Daten, sowie SOAP basierend auf HTTP und XML) ermöglicht OPC-UA auch die semantische Beschreibung der verfügbaren Daten in maschinenlesbarer Form. Das OPC-UA Datenmodell besteht im Wesentlichen aus Knoten (Nodes), vergleichbar mit Objekten aus der Objektorientierten Programmierung, welche Attribute und Methoden besitzen, die gelesen und geschrieben, bzw. aufgerufen werden können. Knoten können sowohl Nutzdaten (z. B. Prozessvariablen), als auch Metadaten (Typinformation) beinhalten. Außerdem werden Events unterstützt. Für verschiedene Anwendungen gibt es standardisierte Datenmodelle. OPC-UA definiert zahlreiche Sicherheitsmechanismen, u. a. Autorisierung und Authentifizierung, sowie kryptographische Signierung, bzw. Verschlüsselung der Daten. Verschiedene Implementierungen, sowohl kommerziell, als auch open source, sind verfügbar. Die Rolle von OPC-UA kann eher im Edge-Bereich, in der Kommunikation mit Automatisierungssystemen, gesehen werden, weniger als universeller Kommunikationsmechanismus für die Kommunikation unter Fog Nodes.

OSGi

OSGi [6] ist eine offene Spezifikation für eine Java-basierte dynamische Softwareplattform basierend auf einem Komponenten und Services-Modell. Von der Grundidee eignet sich das OSGi Architekturmodell sehr gut für die Umsetzung von Edge und Fog Nodes, speziell da es dynamisches Deployment, sowie Upgrades von

Softwarekomponenten unterstützt. Eine wesentliche Einschränkung ist allerdings die Fixierung auf das Java Ecosystem. Allerdings kann das Komponenten- und Services-Modell auch in anderen Programmiersprachen umgesetzt werden.

Eine Open Source IoT Software Plattform für Edge und Fog Computing

macchina.io [7] ist ein Softwarebaukasten zur Entwicklung industrieller Edge- und Fog Computing Applikationen für das Internet der Dinge (IoT). Diese Applikationen laufen auf vernetzten, typischerweise Linux-basierten Geräten und kommunizieren einerseits mit lokal angebundenen Sensoren, Aktoren und anderen Geräten sowie andererseits auch mit anderen Fog Nodes, bzw. Cloud Applikationen. Zu diesem Zweck bietet macchina.io eine Vielzahl an Softwaremodulen die flexibel kombiniert werden können. So werden z. B. verschiedene Kommunikationsprotokolle und -technologien aus dem IoT und Automatisierungsbereich unterstützt, wie z. B. Bluetooth LE, 6LoWPAN, Modbus oder OPC-UA, aber auch Internet Technologien wie HTTP, REST und MQTT.

macchina.io unterstützt mehrere Programmiersprachen, bzw. Programmiermodelle. Als Programmiersprachen werden derzeit C++ und JavaScript unterstützt. C++ wird vornehmlich für low-level Code (z. B. Geräteanbindungen und Kommunikations-protokolle) verwendet, während JavaScript die Entwicklung von Applikationslogik und Workflows stark vereinfacht. Zusätzlich steht eine Dataflow Engine mit einem graphischen, web-basiertem Editor zur Verfügung um einfache Applikationen auch ohne Schreiben von Code realisieren zu können. Die Integration weiterer Programmiersprachen wäre ebenfalls möglich.

Ein in macchina.io integrierter Web Server mit diversen Web-Applikationen (z. B. Bundle Management, Console, JavaScript Playground) ermöglicht speziell während der Entwicklungsphase eine vereinfachte Interaktion mit dem Gerät. Der Web Server ermöglicht auch die Implementierung von REST APIs.

macchina.io basiert auf einer an OSGi angelehnten komponentenbasierten, modularen und dynamischen Architektur (Abbildung 3) deren Kernkonzepte *Bundles* und *Services* sind, implementiert im *Open Service Platform* Framework. Ein Bundle ist ein Deployment-Container für Software-Komponenten, die ausführbaren Code (C++ Shared Libraries, auch für mehrere Architekturen gleichzeitig, oder JavaScript), diverse Ressourcen (z. B. statische HTML Seiten, Stylesheets, Bilder) und Konfigurationsdateien beinhalten, realisiert als Zip Datei mit genau definierter Verzeichnis- und Metadatenstruktur. Bundles können untereinander Abhängigkeiten definieren und haben einen definierten Lebenszyklus, der dynamisch zur Laufzeit gesteuert werden kann. So können jederzeit neue Bundles im System installiert werden, bestehende Bundles gestartet, gestoppt oder entfernt werden, oder auch durch neuere Versionen ersetzt werden. Bundles können kryptographisch signiert werden um den Erzeuger eindeutig identifizieren und Manipulationen ausschließen zu können.

Abbildung 3: macchina.io Architektur

macchina.io selbst besteht aus über 60 Bundles, welche flexibel kombiniert werden können. Auch der Web Server oder das JavaScript Laufzeitsystem sind in verschiedenen Bundles implementiert und können somit weggelassen oder ersetzt werden. Bundles können entweder als Teil des Hauptprozesses laufen, auf mehrere Prozesse aufgeteilt werden, oder in eingeschränkten und abgeschotteten „Sandbox“ Prozessen laufen.

Ein Bundle kann Dienste (Services) bereitstellen, die von anderen Bundles über eine *Service Registry* gefunden und verwendet werden können. Die Suche nach Services erfolgt entweder über den eindeutigen Namen (falls ein spezifisches, bekanntes Service gefunden werden soll), oder über Eigenschaften. So kann z. B. nach allen verfügbaren Temperatursensoren (Sensoren werden als Services abgebildet) gesucht werden. Services werden als C++ Klassen implementiert. Ein in macchina.io integriertes *Remoting* Framework ermöglicht es, diese Services z. B. aus JavaScript Code, aus anderen (Sandbox-)Prozessen (IPC), oder auch remote aufzurufen. Die Vergabe und Kontroller detaillierter Zugriffsrechte ist möglich. Dies ermöglicht die Umsetzung von Microservice-Architekturen.

Mit macchina.io als Edge Computing Plattform wurden bereits mehrere Projekte erfolgreich umgesetzt, z. B. im Automobilbereich (Telematik) oder in der Gebäude-automatisierung. Aber auch der Einstieg ins Fog Computing ist mit macchina.io einfach machbar. Läuft macchina.io auf den einzelnen Fog Nodes, so können Services, Workflows und Applikationen als Bundles auf die zur Verfügung stehenden Nodes verteilt werden.

Zusammenfassung

Edge und Fog Computing werden die Architektur von industriellen IoT Systemen und Anwendungen maßgeblich prägen. Sie ermöglichen es, die „Intelligenz“ von verteilten Systemen aus der Cloud heraus wieder näher zu den eigentlichen „Dingen“

zu bringen und damit die Sicherheit, Leistungsfähigkeit, Skalierbarkeit, Flexibilität, Agilität und Autonomie von IoT Systemen zu verbessern. Bereits jetzt steht eine Vielzahl von Technologien zur Verfügung, um Edge und Fog Computing Architekturen umsetzen zu können. Vom Open Fog Consortium wurde eine Referenzarchitektur für Fog Computing entwickelt; weitere Standards in diesem Bereich werden folgen.

Referenzen

[1] https://www.openfogconsortium.org/ra/

[2] https://swagger.io/specification/

[3] http://mqtt.org

[4] http://portals.omg.org/dds

[5] https://opcfoundation.org

[6] https://www.osgi.org

[7] https://macchina.io

Autor

Günter Obiltschnig ist Gründer der Open Source Projekte macchina.io und POCO C++ Libraries, sowie Geschäftsführer der Applied Informatics GmbH, einem Software-Unternehmen spezialisiert auf Tools und Dienstleistungen rund um das Internet der Dinge. Er verfügt über mehr als 20 Jahre Erfahrung in der Entwicklung von Software für verschiedenste Systeme - von verteilten Unternehmens-applikationen bis zu Embedded Systemen und hält regelmäßig Vorträge auf Fachkonferenzen.