HOCHSCHULE FURTWANGEN UNIVERSITY HFU

Konzeption und Implementierung einer plattformübergreifenden prototypischen Anwendung zum Abruf, Visualisieren und Modifizieren von Daten mittels der in der Industrie 4.0 bekannten OPC Unified Architecture unter Berücksichtigung etablierter Geräteintegrationsstandards am Beispiel von Field Device Tool

Thesis

zur Erlangung des Grades

Bachelor of Science

im Studiengang Wirtschaftsinformatik

an der Fakultät Wirtschaftsinformatik

der Hochschule Furtwangen University

vorgelegt von

Tobias Straub

Referenten:	Prof. Dr. Oliver Taminé
	Dipl.-Ing. Peter Zimmermann
Eingereicht am:	27. August 2016

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Thesis selbständig und ohne unzulässige fremde Hilfe angefertigt habe.

Die verwendeten Quellen sind vollständig zitiert.

Ort, Datum	Unterschrift

Abstract

Die vorliegende Arbeit gibt industriellen Entscheidern, Softwareentwicklern, Geräteherstellern und Standardisierungsgremien einen Überblick über die Problematiken bei der Entwicklung plattformübergreifender Anwendungen zum Abruf, zur Visualisierung und zur Modifizierung heterogener Informationen von unterschiedlichen Geräten über standardisierte Kommunikationstechnologien unter mit Einbezug etablierter Geräteintegrationstechnologien. Zum Aufzeigen der Problematiken wurden exemplarisch die Technologien Open Platform Communications Unified Architecture für den standardisierten Kommunikationsaustausch und Field Device Tool als etablierte Geräteintegrationstechnologie zur Umsetzung der prototypischen Implementierung verwendet. Die wichtigsten Erkenntnisse dieser Arbeit sind neben der Notwendigkeit eines auf plattformübergreifenden Plattformen lauffähigen Toolkits, die Erfordernis zur dynamischen Visualisierung der Daten asymmetrischen Typs, die aufgrund der Informationstypenvielfalt des Open Platform Communications Unified Architecture Standards bestehen. Weiterhin zeigen die Ergebnisse dieser Arbeit die Möglichkeit zur Integration etablierter Geräteintegrationsstandards, sofern diese Integration von den Standardisierungsgremien der etablierten Technologien korrekt realisiert wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API Programmierschnittstelle.
siehe auch Glossar: Programmier- schnittstelle.

CPS Cyber-physisches System.
siehe auch Glossar: Cyber-physisches System.

CSS Cascading Style Sheets.
siehe auch Glossar: Cascading Style Sheets.

DTM Device Type Manager.
siehe auch Glossar: Device Type Manager.

DVD Digital Versatile Disc.

ER Entity-Relationship.
siehe auch Glossar: Entity-Relationship.

ERP Enterprise-Resource-Planning.
siehe auch Glossar: Enterprise-Resource-Planning.

FDT Field Device Tool.
siehe auch Glossar: Field Device Tool.

GDS Global Discovery Server.
siehe auch Glossar: Global Discovery Server.

HTML Hypertext Markup Language.
siehe auch Glossar: Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.
siehe auch Glossar: Hypertext Transfer Protocol.

IOSB-INA Fraunhofer-Anwendungszentrum Industrial Automation.

LDS Local Discovery Server.
siehe auch Glossar: Local Discovery Server.

MES Manufacturing Execution System.
siehe auch Glossar: Manufacturing Execution System.

MVVM Model View ViewModel.
siehe auch Glossar: Model View ViewModel.

PDF Portable Document Format.

REST Representational state transfer.
siehe auch Glossar: Representational state transfer.

SDK Software Development Kit.
siehe auch Glossar: Software Development Kit.

SOA serviceorientierten Architektur.
siehe auch Glossar: serviceorientierte Architektur.

SOAP Simple Object Access Protocol.
siehe auch Glossar: Simple Object Access Protocol.

TCP Transmission Control Protocol.
siehe auch Glossar: Transmission Control Protocol.

UA Open Platform Communications Unified Architecture.
siehe auch Glossar: Open Platform Communications Unified Architecture.

UWP Universal Windows Platform.
siehe auch Glossar: Universal Windows Platform.

XAML Extensible Application Markup Language.
siehe auch Glossar: Extensible Application Markup Language.

XML Extensible Markup Language.
siehe auch Glossar: Extensible Markup Language.

Glossar

Programmierschnittstelle	Ermöglicht die Kommunikation zwischen zwei Systemen, meist im Zusammenhang durch Aufruf von Diensten.
Cyber-physisches System	Cluster von informationstechnischen und mechanischen sowie elektrischen Komponenten.
Cascading Style Sheets	Formale Sprache zur Beschreibung grafischer Benutzeroberflächen.
Device Type Manager	Softwarekomponente innerhalb der Field Device Tool Technologie zur Repräsentation eines physikalischen Gerätes.
Entity-Relationship	Diagrammsprache zur Datenbankmodellierung.
Enterprise-Resource-Planning	Planungs- und Steuerungssystem von Organisationstätigkeiten und Ressourcen.
Field Device Tool	Technologie die eine vom Gerätehersteller unabhängige Parametrisierung von Geräten über einheitliche Schnittstellen und eine einheitliche grafische Benutzeroberfläche ermöglicht.
Global Discovery Server	Server, welcher Informationen über im Netzwerk existierende Open Platform Communications Unified Architecture Server enthält [1].

Hypertext Markup Language	Auszeichnungssprache zur Auszeichnung digitaler Dokumente.
Hypertext Transfer Protocol	Protokoll zur Datenübertragung.
Local Discovery Server	Server, welcher Informationen über im Netzwerk existierende Open Platform Communications Unified Architecture Server enthält, und selbst die gleiche Workstation als Laufzeitumgebung nutzt, wie die Server zu denen dieser Informationen bereitstellt [1].
Manufacturing Execution System	System zur Planung und Durchführung der Produktion durch direkte Anbindung an Systeme der Prozessautomatisierung.
Model View ViewModel	Entwurfsmuster zur Trennung von Präsentation, der Logik der Präsentation sowie der Anwendungslogik.
Representational state transfer	Programmierparadigma für Dienste.
Software Development Kit	Anwendungen die es erlauben eine neue Anwendung zu entwickeln meist durch Einbindung.
Serviceorientierte Architektur	Architekturentwurfsmuster zur Interaktionsstrukturierung von Diensten.
Simple Object Access Protocol	Industrieller Protokollstandard zum Datenaustausch und Funktionalitätenaufruf.

Transmission Control Protocol	Protokoll zur Datenübertragung der Internetprotokollfamilie.
Open Platform Communications Unified Architecture	Technologie die ein Datenaustausch auf vertikaler Unternehmensebene unabhängig der Daten und der dahinter liegenden Sende-/Empfangskomponenten ermöglicht.
Universal Windows Platform	Laufzeitumgebung für Anwendungen, die unter Windows 10 ausgeführt werden können.
Extensible Application Markup Language	Beschreibungssprache für grafische Benutzeroberflächen.
Extensible Markup Language	Auszeichnungssprache zur Strukturierung von Dokumenten.
Künstliche Intelligenz	Die Fähigkeit digitaler Systeme, zur Lösung von Aufgaben die eine höhere intellektuelle Verarbeitungsfähigkeit erfordern, die derzeit von Menschen besser verstanden und gelöst werden können [2].
Industrie 4.0	Beschreibt Technologien und Konzepte, die zur Wertschöpfungsoptimierung beitragen, durch den Einsatz von intelligenten Informationssystemen.
Wertschöpfungskette	Abstrakte Beschreibung aller primären übergeordneten Prozesse in einem Unternehmen [3].
Wertschöpfung	Realisierter Gewinn von Unternehmen durch die Erbringung der Transformationsleistung von Gütern zu Produkten.

Geräteintegration Die Eingliederung von heterogenen und homogenen Geräten in eine IT-Landschaft, welche über eine Technologie innerhalb der IT-Landschaft, einen heterogenen Zugriff auf die eingegliederten Geräte ermöglicht.

Endgerät Gerät, welches mit einem Netzwerk verbunden ist.

Technologie Die Lehre des Wissens sowie der Betrachtungsweisen zur Realisierung der Technik [4].

Plattform- unabhängigkeit Die Ausführung einer Anwendung auf verschiedenen Laufzeitumgebungen mit möglichen homogenen Eigenschaften dieser Plattformen, wobei die Anwendung im Entwicklungsprozess auf einer einzigen Quellcodebasis realisiert wurde.

Xamarin Entwicklungsplattform zur Entwicklung von nativen iOS, Android und Windows Anwendungen unter Nutzung einer einzigen Quellcodebasis durch die Transpiler und Programmierschnittstellen Abstraktion von Xamarin [5, 6].

Prototyp Eine lauffähige Anwendung oder Teilmenge der Anwendung zur Untersuchung verschiedenster vorher definierter Analyseaspekte, die nach Abschluss zur vollständigen beziehungsweise Richtlinien-weisenden Implementierung der Anwendung führen.

Plattform Laufzeitumgebung für Anwendungen.

Prototyp Lauffähiger Quellcodeabschnitt einer Zielanwendungsdefinition.

Internet der Dinge	Vernetzung von Menschen, Anwendungen, Informationen, Sensoren, Messwerte der Sensoren, Geräte sowie alle möglichen vorstellbaren Gegenstände [7].
Mechanik	Bewegung von Körpern bei Einfluss externer Kräfte.
Mechatronik	Gemeinsames Wirken von Mechanik und Elektronik.
Adaptronik	Beschreibt die Adaption durch Elektronik.
Internetadresse	Eine Adresse zur Identifizierung einer Ressource im Internet.
Semantik	Die Bedeutung, die einer Zeichenkette zugemessen wird.
Liefertreue	Die Einhaltung eines vorher definierten Liefertermins.
Produktionsstoff	Ein Rohstoff, Hilfsstoff oder Betriebsstoff, der zur Produktion des Endproduktes verwendet wird.
Digitale Transformation	Die digitale Transformation im Kontext der Industrie.
Rüsten	Einrichten von Betriebsmitteln zur Durchführung einer definierten Tätigkeit.
Horizontale Integration	Integration von IT-Systemen entlang der Wertschöpfungskette.

Vertikale Integration	Integration von IT-Systemen auf unterschiedlichen Unternehmenshierarchieebenen.
Operative Informationssysteme	Software, die Mitarbeiter in operativen Ebenen einer Organisation hinsichtlich operativen Tätigkeiten unterstützt.
Planungs- und Kontrollsysteme	Software, welche Mitarbeiter bei Planungs- und Kontrollaufgaben unterstützt.
Automatisierungspyramide	Grafische Abbildung einer Pyramide zur Einordnung verschiedener Technologien der Automatisierungsindustrie.
Sensor	Technisches Bauteil zur Erfassung von Daten, die je nach Sensor unterschiedlich sind.
Aktor	Umwandlung von Signalen nach physikalischen Größen.
Feldgerät	Sensoren oder Aktoren innerhalb der Fabrik, in der diese verwendet werden.
Dienst	Bündelung von Funktionalitäten, die durch eine definierte Schnittstelle aufgerufen werden können.
Protokoll	Regelwerk zur Datenübertragung zwischen zwei digitalen Systemen.
Electronic Device Description Language	Beschreibungssprache zur Verwendung und Parametrisierung von Geräten.

Field Device Integration	Technologie zur Parametrisierung von Geräten verschiedener Gerätehersteller auf horizontaler Unternehmensebene.
IO-Link	System zur Bindung von Sensoren und Aktoren durch Standardisierung von Anschlussdaten und einem entsprechenden Datenübertragungsprotokoll.
Peripherie	Geräte, die Bestandteil eines Computersystems sind, durch Kommunikation mit der zentralen Einheit des Computersystems.
Wireless Local Area Network	Funknetz zur Datenübertragung.
Global System for Mobile Communications	Standard zur Datenübertragung von Telefonie und Nachrichten.
General Packet Radio Service	Technologie zur Datenübertragung in Mobilfunknetzen.
Laufzeitumgebung	Ermöglicht die Ausführung von Anwendungen durch Bereitstellung von Funktionalitäten, primär zum Zugriff auf Peripherie.
Hypertext Transfer Protocol Secure	Protokoll zum verschlüsselten Datentransport.
Web Services Description Language	Beschreibungssprache für Dienste.

Supervisory Control and Data Acquisition	System zur Fernüberwachung sowie Fernsteuerung von industriellen Prozessen.
Anwendungsfall	Beschreibt eine Funktionalität des zu entwickelnden Systems.
Stack	Teilsoftwarekomponente, die große Teile der technischen Umsetzung des Open Platform Communications Unified Architecture Standards abstrahiert.
Administrative Domäne	Menge von Netzwerken und Netzwerkkomponenten, die ein Unternehmen verwaltet.
Endpunkt	Bezeichnet die Adresse eines Dienstes.
Discovery	Auffinden von definierten UA Servern innerhalb einer lokalen Maschine, eines Netzwerksegments oder der gesamten administrativen Domäne.
Objektorientiertes Programmierparadigma	Beschreibt die Kapselung von Daten in Objekten und die Trennung dieser Objektdaten von den auf die Objekte anzuwendenden Routinen.
Metadaten	Daten, die Eigenschaften für welche die Eigenschaften beschrieben werden, enthalten.
Stillstandskosten	Durch Pausieren des Produktionsprozesses weiterhin entstehende Kosten und die durch das pausieren des Produktionsprozesses zu entstehenden Kosten.
Informationsmodell	Schema zum strukturierten Ablegen von Dateninhalten.

FDT Group	Standardisierungsgremium zur Standardisierung der Field Device Tool Technologie.
OPC Foundation	Standardisierungsgremium zur Standardisierung verschiedenster Technologien, unter anderem der Open Platform Communications Unified Architecture.
JavaScript	Skriptsprache mit breitem Anwendungsgebiet.
Toolkit	Vereinigung von Stack und Software Development Kit zur gemeinsamen Aufgabenbearbeitung.
Visual Studio	Integrierte Entwicklungsumgebung.
IP-Adresse	Eindeutige Identifikationsadresse zum Senden und Empfangen von Datenpaketen, für an der Netzkommunikation teilnehmender Geräte.
Hostname	Eindeutige Identifikationsbezeichnung eines an der Netzkommunikation teilnehmender Geräte.
Ethernet	Technologie zum Datenaustausch.
Port	Ermöglicht die Zuordnung von Verbindungen zu Anwendungen auf Betriebssystemebene.
Lose Kopplung	Geringe Abhängigkeit von Softwarekomponenten untereinander.
Datenbank	Strukturierte Persistierung meist großer Datenmengen.

Relationale Datenbank	Strukturierte Persistierung meist großer Datenmengen in Tabellenform.
Schlüssel/Wert Speicher	Persistierung unstrukturierter meist kleiner Datenmengen.
SQLite	Relationale Datenbank die als Programmbibliothek bereitgestellt wird.
Objektrelationale Abbildung	Bezeichnet eine Variante, in der die Umwandlung von Daten zur Persistierung oder zum Konsum einer nicht objektorientierten Datenbank in Objekten abstrahiert wird.
Dependency Injection	Entwurfsmuster zur Verwaltung von Abhängigkeiten zwischen Softwarekomponenten zur Laufzeit.
NuGet	Paketverwaltung für die Softwareentwicklung unter .NET.
Autofac	Programmbibliothek zur Umsetzung des Dependency Injection Entwurfsmuster.
Build Action	Eigenschaft, die als Indikator für den Kompiler zur Abbildung verschiedenster Kompilierungsmöglichkeiten dient.
Referen- zarchitekturmodell Industrie 4.0	Modell, das Unternehmen als Referenz für die Entwicklung zukünftiger Produkte und Geschäftsmodelle dienen soll.
x509-Zertifikat	Standard, der die Infrastruktur beschreibt, die zur Erstellung von digitalen Zertifikaten mittels einer Schlüssel/Wert-Struktur führt.

Kerberos	Protokoll zur Authentifizierung in verteilten Netzen.
Secure Sockets Layer	Protokoll zur sicheren Datenübertragung.
Speicher- programmierbare Steuerung	Programmierbares Gerät zur Steuerung einer Anlage.
Feldbus	Verbindet Feldgeräte mit Automatisierungssystemen zur Kommunikation.

1. Einleitung

Nach dem exzessiven Bevölkerungswachstum Mitte des 18. Jahrhunderts stellte der schottische Pastor Thomas Robert Malthus eine im Dokument *An Essay on the Principle of Population* [8] niedergeschriebene Theorie auf. Dieser Theorie zufolge lassen sich mit den damalig zur Verfügung stehenden Produktionsmethoden, die Nahrungsmittel nur in arithmetischer Reihe vermehren. Malthus kam allerdings zur Erkenntnis, dass die Bevölkerung nicht arithmetisch, sondern progressiv ansteigt. Um Produktionssteigerungen zu ermöglichen, die der Problematik entgegenwirken, mussten neue Produktionsmethoden geschaffen werden. James Watt machte mit der Dampfmaschine die hierzu entscheidende Erfindung. [9] Wie Abbildung 1.1 zeigt, wurde die Industrie in den folgenden Jahrzehnten durch weitere Revolutionen geprägt.

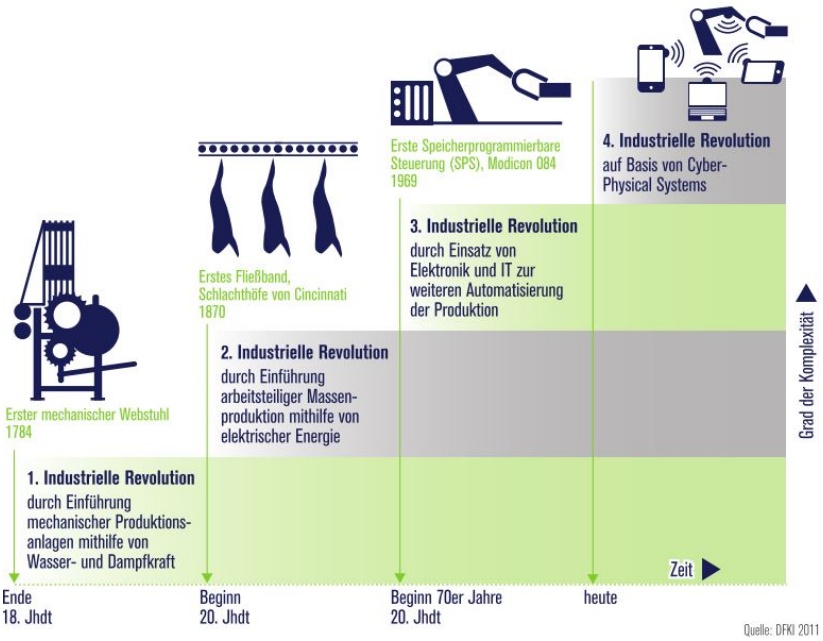

Abbildung 1.1.: Revolutionsstufen in der industriellen Produktion der vergangenen Jahre nach Grad der Komplexität [10]

Nach der Entdeckung von Dampf und Wasserkraft folgte die elektrische Energie und Massenfertigung. Mit Beginn der siebziger Jahre kam schließlich der Einsatz von Informationstechnologie. [11] Derzeit befinden wir uns im Wandel vom Einsatz unintelligenter Produktionstechnologie hin zu einer dezentralen und selbst organisierenden Struktur, die über den gesamten Lebenszyklus von Produkten auf Basis künstlicher Intelligenz reicht. [11, 12, 13] Der Wandel mit den Technologien und Konzepten, zur Realisierung wird als die vierte industrielle Revolution bezeichnet. Hierzu hat sich das Leitwort *Industrie 4.0* etabliert. [11]

Die durch künstliche Intelligenz unterstützten Überwachungs- und Entscheidungsprozesse ermöglichen es Unternehmen die gesamte Wertschöpfungskette in Echtzeit zu steuern und zu optimieren. [14] Zur Realisierung der in der Vision Industrie 4.0 angestrebten Ziele mit Konzepten und Technologien, insbesondere mit dem Kernkonzept der künstlichen Intelligenz bedarf es einer komplexen Kommunikation die eine Totalvernetzung der Produktionskette und der gesamten Wertschöpfungskette ermöglicht. Die Umsetzung einer Totalvernetzung erfordert, dass Produktionskomponenten als auch die Produkte selbst in die Kommunikation mit IT-Systemen auf Unternehmensbasis integriert werden. [11, 15, 16] Bei der Einbettung der Produktionskomponenten und Produkte in die Gesamtkommunikation zeigt sich die Schwierigkeit mit der Heterogenität der an der Wertschöpfung beteiligten Produktionskomponenten. Die Heterogenität wird durch den Einsatz verschiedener Geräte charakterisiert, meist verschiedener Hersteller, die unter Umständen über verschiedene Kommunikationstechnologien kommunizieren.

Gegenstand dieser Arbeit ist die Beantwortung folgender Forschungsfrage: Wie lassen sich im Kontext von Industrie 4.0 durch standardisierte Kommunikationstechnologien die verschiedenen spezifischen Daten eines Gerätes unabhängig vom Hersteller des Gerätes und unabhängig von der eigentlichen Information sowie unter Berücksichtigung etablierter Geräteintegrationsstandards in einer von der Plattform und Endgerät unabhängigen Anwendung abrufen, visualisieren und modifizieren?

Dem Titel dieser Arbeit entsprechend basiert die Problemlösung auf den Tech-

nologien Open Platform Communications Unified Architecture (UA) und Field Device Tool (FDT). Die Betrachtung weiterer möglicherweise geeigneter Geräteintegrationstechnologien ist nicht Gegenstand dieser Arbeit. Als Grund sei hierfür neben den durch das Unternehmen M&M Software GmbH gestellten Randbedingungen die besondere Aktualität und Verbreitung beider Technologien im Kontext von Industrie 4.0 anzuführen. Die zur Auslieferung von entsprechenden Daten notwendige UA Server werden durch das Unternehmen M&M Software GmbH bereitgestellt. Ebenfalls als Randbedingung des erwähnten Unternehmens und aufgrund besonderer Aktualität im Bereich der Entwicklung von plattformübergreifenden Anwendungen definiert ist die Verwendung von Xamarin. Mit Xamarin soll die plattformübergreifende Umsetzung der prototypischen Implementierung zur Erbringung des Nachweises von Plattform und Endgerät unabhängigen Abrufs sowie Visualisierung und Modifizierung von den Technologien gelieferten Daten geschaffen werden.

Der Aufbau dieser Arbeit gliedert sich neben der Einleitung in fünf Hauptkapitel. Das Kapitel 2 kann als Hinführung zur Forschungsfrage betrachtet werden. Es werden die besonderen Merkmale der vierten industriellen Revolution aufgezeigt, die zu einer *Kommunikationsproblematik* führen. Aus dieser Kommunikationsproblematik resultiert wiederum die Forschungsfrage der Arbeit. Kapitel 3 analysiert unter anderem die Möglichkeiten der Technologien, die in der Kontextdarlegung als theoretische Lösungen für die Kommunikationsproblematik beschriebenen wurden. In der Analyse werden ebenfalls die in der Forschungsfrage definierten Bedingungen zur Unterstützung etablierter Geräteintegrationstechnologien und die Realisierung des Datenaustausches unabhängig der Plattform beachtet. Weiterhin wird der Systemkontext definiert, der sich durch die Analyse der Technologien ergibt. Aus den Möglichkeiten der Technologien lassen sich Minimalanforderungen ableiten, die zur Beantwortung der Forschungsfrage führen. Im Mittelpunkt des Kapitels 4 stehen die Konzepte, Strategien und Entscheidungen, die als Grundlage für die Realisierung des Prototyps in Kapitel 5 führen. Das Kapitel 5 behandelt dementsprechend das Design und die Implementierung, zu den in Kapitel 4 getroffenen Entwurfsentscheidungen, die den Kern der Anwendung ergeben. Kapitel 6 resümiert die Arbeit und bildet somit die Schlussbetrachtung.

2. Kontextbetrachtung

Industrie 4.0 kann formal als Ansammlung von Technologien und Konzepten verstanden werden, welche die Produktion von Produktvielfalten bis hin zur Losgröße eins unter Bedingung von Großserienproduktionen ermöglichen, um so gleichwohl eine massive Optimierung der Wertschöpfung zu erreichen. Hierdurch entsteht eine hohe Anzahl von Daten, welche im gesamten Wertschöpfungsprozess distribuiert werden müssen. Dies skizziert den Kontext für die zu beantwortende Forschungsfrage, welcher in den nächsten Unterkapiteln genauer erläutert wird.

2.1. Industrie 4.0

Unter dem Leitwort Industrie 4.0 hält das Internet der Dinge Einzug in die Fabrik. [17] Dies bedeutet die Vernetzung von Personen, Dingen und Maschinen mit dem primären Ziel der Produktindividualisierung unter Bedingung einer Großserienproduktion. [14, 15, 18] Der Konsumentenwunsch nach maßgeschneiderten Produkten der Losgröße eins gewann in den letzten Jahren stark an Bedeutung. [14, 18] Trotz hohen Bestrebens war die Erfüllung dieses Konsumentenwunschs aufgrund mangelnder Rentabilität, fehlender Wirtschaftlichkeit und nicht zu bewerkstelligten Steigerungen in den Wertschöpfungsanteilen nicht zu realisieren. [10, 15, 18, 19] Mit den in Industrie 4.0 definierten Technologien und Konzepten soll diese Schwierigkeit wegrationalisiert werden. Hierzu tragen vor allen Dingen die immer lernfähiger werdenden Maschinen bei, die als Teil zum Umbruch von unintelligenter Produktionstechnologie zu einer auf Basis von künstlicher Intelligenz dezentralen und selbst organisierten Struktur heranwachsen. [10, 11, 12, 13, 15, 18, 19] Industrie 4.0 transformiert Industrieunternehmen vollständig. [15] Hierdurch verändert sich nicht nur die

am Konsumenten zu erbringenden Dienstleistungen, sondern zusätzlich die Beziehung zwischen Konsument und Unternehmen. [19] Der Konsument wird zukünftig direkt in Geschäfts- und Wertschöpfungsprozesse integriert werden. [14] Der im Kontext der vierten industriellen Revolution besonders stark zu erkennende, kontinuierliche Verbesserungsprozess der Wertschöpfung ist kein neues Phänomen. Abbildung 2.1 zeigt die stetige Verbesserung der Wertschöpfung durch die Integration neuer Funktionen seit Beginn der ersten industriellen Revolution. Aus mechanischen Systemen wurden mechatronische, die durch adaptronische ersetzt wurden. Diese wurden schließlich durch die Bindung von cyber-physischen Systemen ersetzt. [10, 11]

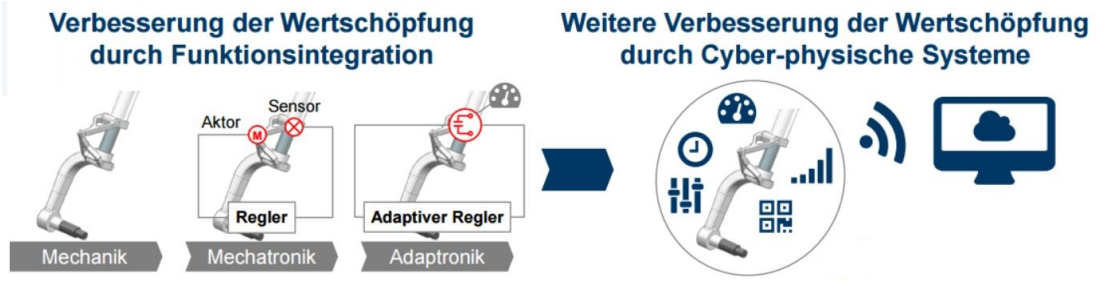

Abbildung 2.1.: Verbesserung der Wertschöpfung durch Funktionsintegration [11]

Cyber-physische Systeme (CPS) ermöglichen die Kopplung realer (physischer) Objekte mit informationsverarbeitenden (virtuellen) Objekten und Prozessen über Informationsnetzwerke wie beispielsweise dem Internet hinweg. [17, 20] Ein CPS ist eines der grundlegenden Konzepte der Industrie 4.0, ist aber nicht speziell durch diese Revolution erschaffen worden. [20] Im Kontext des globalen Internets stellt die Technologie allerdings eine Neuerung für die Kommunikation innerhalb von Industrie 4.0 dar. [17, 20] Vereinfacht ausgedrückt kann ein CPS als ein modernes Steuerungssystem mit eingebettetem Softwaresystem und im Rahmen der Industrie 4.0 mit einer integrierten Internetadresse bezeichnet werden. [11] Durch die Integration solcher Steuerungssysteme durch Eingliederung dieser innerhalb der Produktion ergibt sich ein Netzwerk anpassungsfähiger, sich selbst organisierender Produktionskomponenten. Durch die daraus resultierende hohe Flexibilität der gesamten Wertschöpfungsprozesse wird die angestrebte effiziente und kostengünstige Produktion von Individualprodukten erreicht. [10, 15, 18, 19, 20] Bei zusätzlicher Einbindung intelligenter

Werkstückträger und der Verankerung eines semantischen Produktgedächtnisses auf dem zu entstehenden Produkt wird eine Kommunikation zwischen Produkt und Produktionskomponenten gewährleistet. Entsprechend den lang- oder kurzfristig geplanten im Produktgedächtnis auf dem Produkt persistierten Verarbeitungsschritten können die Produkte durch die Werkstückträger zur entsprechenden Produktionsanlage zur Weiterverarbeitung gefahren werden. Produktionsanlagen können das Produktgedächtnis ebenfalls lesen und die Produktionsschritte umsetzen. Durch diese Konzepte entsteht so ein dezentrales, dynamisches Steuerungsverfahren, bei dem das Produkt den Produktionskomponenten durch Orchestrierung selbiger kommuniziert, wie es produziert werden muss. Dies macht beispielsweise die Wahl zwischen Ressourcenschonung und Liefertreue möglich. Soll das Produkt möglichst ressourcenschonend produziert werden, würden die entsprechend zu Produktionskomponenten gefahren werden, die möglichst wenig CO2-Ausstoß verursachen, aber unter Umständen die Liefertreue behindern. [21]

Aller Voraussicht nach werden bis zum Jahr 2020 deutsche Industrieunternehmen jährlich 40 Milliarden Euro in Industrie 4.0 Lösungen investieren. 85 Prozent aller Unternehmen werden zu diesem Zeitpunkt bereits in allen wichtigen Unternehmensbereichen Industrie 4.0 Lösungen implementiert haben. Kernfähigkeit von Unternehmen bei der digitalen Transformation wird die integrierte Analyse und Nutzung von Daten sein. [15] Die Datensicherheit sowie die Sicherheit der vollständigen IT-Infrastruktur wird eines der größten Hürden beim Wandel darstellen. [19] Die definierten Konzepte transformieren das Unternehmen in eine von Daten getriebene Fabrik. Dieser Datentrieb spiegelt sich im Fabrikgedächtnis wider. Hierbei handelt es sich um die Summe aller Produktgedächtnisse, Maschinengedächtnisse und Informationen der Werker. [21] Festzustellen ist, dass der wesentliche Treiber für den Vormarsch von Industrie 4.0 Lösungen die Konzepte zur Steuerung der vertikalen und horizontalen Wertschöpfungskette sind. Diese Konzepte setzen die Fokussierung auf den effizienten Datenaustausch innerhalb der Wertschöpfungskette, die digitale eindeutige Identifikation der Produkte und die Nutzung von Echtzeitdaten zur Steuerung der Produktion. [15]

2.2. Kommunikationsproblematik

Laut einer Umfrage [10] unter Verbandsmitgliedern der Plattform Industrie 4.0 stellt die Standardisierung eines der größten Hürden bei der Umsetzung der vierten industriellen Revolution dar. Im vorangegangenen Unterkapitel 2.1 beschrieben, stellt die Totalvernetzung das Fundament für alle folgenden Industrie 4.0 Technologien und Konzepte. Somit übernimmt gerade die Standardisierung insbesondere im Rahmen der Kommunikation eine besonders tragende Rolle. Zur Beantwortung der Forschungsfrage ist es von Bedeutung die Schwierigkeiten einer solchen Totalvernetzung in industriellen Wertschöpfungsprozessen zu erkennen und zu verstehen.

Eine standardisierte und durchgängige Kommunikation realisiert die Vision intelligenter Produkte und Produktionskomponenten die eine dezentrale, selbst organisierende und auftragsspezifische Produktion ermöglichen. Die Kommunikation ermächtigt das Produkt den Produktionskomponenten spezifische Anweisungen zu, den für das zur Herstellung des Produkts notwendigen Arbeitsschritten zu geben. Dies Erfordernis zur Implementierung der Vision ergibt sich durch den in Kapitel 2.1 beschriebenen Wunsch zur wirtschaftlich rentablen und variantenreichen Produktion ohne die Notwendigkeit eines manuellen Rüsten der Produktionsanlagen. [11, 13, 19]

Spricht man in Industrie 4.0 von Kommunikation, so muss man zuvor die Integration von Informationssystemen in der Wertschöpfungskette betrachten. Es werden zweierlei Integrationsformen unterschieden: Horizontale Integration und Vertikale Integration. Bei der horizontalen Integration handelt es sich um die Integration von IT-Systemen entlang der Wertschöpfungskette. Von vertikaler Integration hingegen spricht man bei der Integration von IT-Systemen auf unterschiedlichen Unternehmenshierarchieebenen, wie beispielsweise operative Informationssysteme [22] oder Planungs- und Kontrollsysteme [22]. [11, 23] Abbildung 2.2 veranschaulicht die strikte Trennung der beiden Integrationsformen in der sogenannten Automatisierungs- pyramide, welche zur Einordnung von IT-Systemen im Automatisierungsumfeld dient. Auf unterster Ebene der Automatisierungs- pyramide befinden sich Sensoren und Aktoren. Diese werden auch unter dem Begriff *Feldgeräte* zusammengefasst, wodurch diese

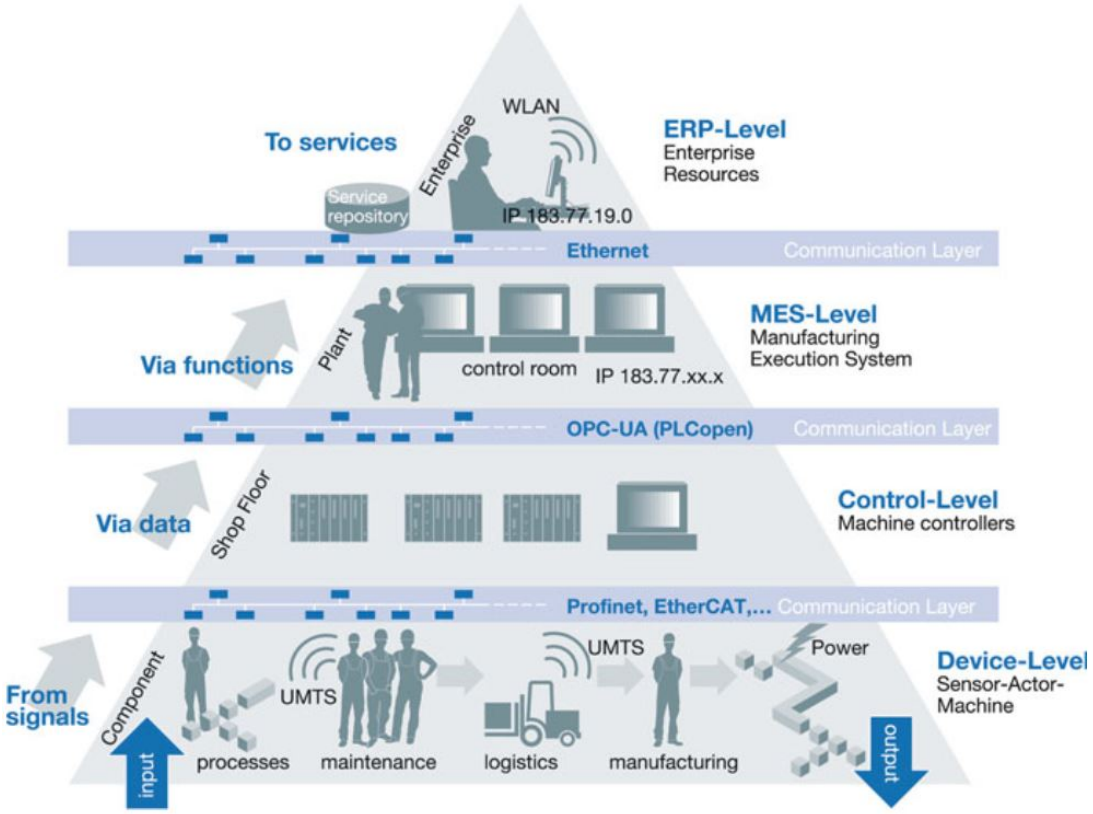

Abbildung 2.2.: Strikte Trennung zwischen horizontaler und vertikaler Integration, dargestellt durch die Automatisierungspyramide [13]

Ebene umgangssprachlich als *Feldebene* bezeichnet wird. Diese Feldgeräte sind mit Steuerungsgeräten verbunden, welche die Gerätesignale in besser verständliche Daten konvertieren. Nach der Konvertierung werden die Daten an die nächsthöhere Ebene der Automatisierungs- pyramide, der Prozessleitebene weitergereicht. Über in sogenannte *Dienste* gebündelte Funktionalitäten können die höheren Ebenen (Manufacturing Execution System (MES) - Enterprise-Resource-Planning (ERP)) mit den darunterliegenden Ebenen kommunizieren. Initiiert wird diese Kommunikation in aller Regel durch ein Endgerät der Unternehmensleitebene. [13] Betrachtet man die Kommunikation zwischen den einzelnen Ebenen der Automatisierungs- pyramide genauer, erkennt man die Problematik zur Umsetzung von Industrie 4.0: Geräte unterschiedlicher horizontaler Ebenen kommunizieren jeweils über uneinheitliche

oft für die Domäne spezifische Protokolle. [16, 24, 25, 26] Die beschriebene Zieldefinition von Industrie 4.0 erfordert aber eine notwendige vertikale Vernetzung über die gesamten Ebenen hinweg. [24, 25] In der Vergangenheit wurden diverse domänenspezifische Protokolle auf horizontaler Ebene eingeführt. Beispiele hierfür sind FDT, Electronic Device Description Language, Field Device Integration oder IO-Link. Sucht man nach standardisierten Protokollen, die eine vertikale Vernetzung ermöglichen, findet man dort nur eine kleine etablierte Menge an Technologien. Es ist festzuhalten, dass obwohl die Interaktion von IT- und Automatisierungswelt in der Vergangenheit eine anspruchsvolle Aufgabe darstellte, eine Interaktion der beiden Welten dennoch nicht völlig visionär ist. Zur Realisierung der Konzepte von Industrie 4.0 muss die strikte Trennung der einzelnen Ebenen in der Automatisierungs- pyramide aber weiter aufweichen und sich vermischen müssen. [13] Erst dadurch ist die intelligente Vernetzung ohne Begrenzung von Kommunikationsintegrationsgrenzen möglich und jedes Gerät und jeder Dienst kann eine Kommunikation mit beliebigen anderen Geräten und Dienste über standardisierte Schnittstellen initiieren. [13, 26] Die folgende Abbildung 2.3 verdeutlicht die Wandlung von einer hierarchischen Kommunikationsstruktur hin zu einem verteilten Netz, das auf cyber-physischen Diensten basiert. [20]

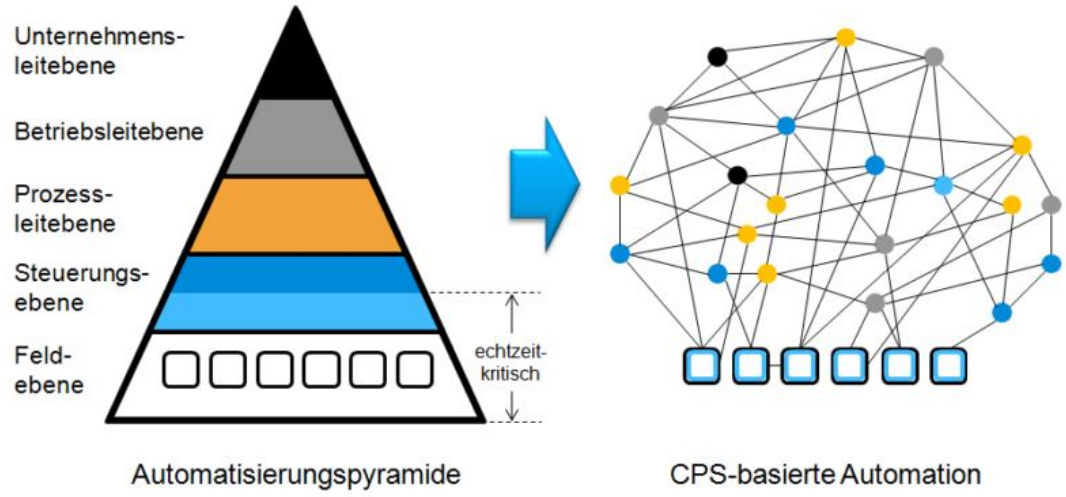

Abbildung 2.3.: Wandlung der hierarchischen Kommunikationsstruktur zu einem verteilten Netz von cyber-physisch basierten Diensten [20]

Die Frage, die sich dem aufmerksamen Leser stellen sollte, ist die Frage nach dem *Wie*. Wie lässt sich diese strikte Trennung der beiden Integrationsformen von der hierarchischen Struktur, welche die Automatisierungs- pyramide in Abbildung 2.2 illustriert, hin zu einem Netzwerk von cyber-physisch basierten Diensten wie Abbildung 2.3 zeigt, ermöglichen? Dies ist zugleich die Ausgangsfrage, die als Grundlage zur Beantwortung der Forschungsfrage dient. Bevor in Abhängigkeit zur Forschungsfrage Daten abgerufen, visualisiert und aktualisiert werden können, müssen die entsprechenden Daten erst einmal in einem konsumierbaren und einheitlichen Datenmodell festgehalten werden, mit dessen ein Abruf und entsprechend eine Modifizierung der Daten ermöglicht werden kann. - Beobachter informationstechnischer Verarbeitungsprozesse würden erklären, dass bereits etablierte Technologien und Peripherien existieren, die eine Vernetzung hin zum von Diensten getriebenen Netz über bisherige Kommunikationsgrenzen hinweg ermöglichen. Exemplarisch hierfür sind Übertragungsperipherien wie Kabel, Wireless Local Area Network, Global System for Mobile Communications oder General Packet Radio Service und Protokolle wie Transmission Control Protocol (TCP) oder Hypertext Transfer Protocol (HTTP). Alle Technologien und Peripherien sind international standardisiert und somit grundsätzlich für die von Standards getriebenen vierten industriellen Revolution von Bedeutung. Für die von Industrie 4.0 definierten Konnektivitätsanforderungen von Geräten und Dienste reichen diese alleine nicht aus. So stellt Industrie 4.0 als eine der wichtigsten Anforderungen zur Konnektivität die semantische Interoperabilität der Technologien, gekoppelt mit Sicherheits- und Zugriffsmechanismen voraus. Diese definierte Anforderung impliziert die Notwendigkeit einer vollständigen Plattform für den Informations- und Kommunikationsaustausch zwischen Geräten und Systemen. [27] Spricht man von Plattform, wird damit eine Laufzeitumgebung für darauf aufsetzende Anwendungen definiert. [28] Die bisher erwähnten Technologien können dies nicht erfüllen, da diese lediglich die Transportschicht der Daten bedienen. In der IT-Landschaft existieren dennoch bereits einfache Plattformen zum Datenaustausch, die aber die für Industrie 4.0 Entscheidens kritische Anforderung der Interoperabilität nicht erfüllen. Als Beispiel sei hierfür Representational state transfer (REST) mit dem Transport über die Protokolle HTTP/Hypertext Transfer Protocol Secure und der Beschreibung der Dienstschnittstellen über Web

Services Description Language zu erwähnen. Wurden in der Vergangenheit Gerätehersteller gebundene Webdienste und Schnittstellen über beispielsweise REST eingesetzt, musste mit jeder Erweiterung der Gerätefunktionalität eine Welle von Softwareanpassungen mit üblichen Softwaretests folgen. Zur Lösung der beschriebenen Problematiken haben sich mehr als 470 Unternehmen [27] unter dem Dach des Standardisierungsgremiums OPC Foundation zusammengeschlossen. Das angestrebte Ziel war die Verabschiedung eines Standards, der die erwähnten Problematiken lösen kann. [13] Resultat ist ein leistungsstarker und für die Automatisierungsbranche ein derzeit zu etablierender Standard für den Daten- und Informationsaustausch. [13, 16, 29] Dieser ermöglicht ein Datenaustausch von völliger Unabhängigkeit des Geräteherstellers, Betriebssystems sowie Hierarchie und Topologie der unterliegenden Systeme. Grundlegendes Modellkonzept für den Datenaustausch ist dabei das im UA Standard definierte Informationsmodell. [13] Gerätehersteller bilden hierbei die Informationsarten von dessen hergestellten Geräten in das UA Informationsmodell ab. [30] UA Anwendungen können die im Standardinformationsmodell abgebildeten Daten der Geräte dann entsprechend über den von im UA Standard festgelegten Satz von 37 Dienstschnittstellen aus dem Informationsmodell heraus abrufen und modifizieren. [13, 24, 25] Die Technologie unterscheidet somit explizit zwischen Mechanismen für den Informationsaustausch und den auszutauschenden Daten als solche. [13, 31]

Zur Realisierung der durchgängigen Industrie 4.0 Kommunikation wird der UA Standard auf allen Ebenen der traditionellen Automatisierungs- pyramide als Schlüsseltechnologie anzutreffen sein. [29, 32] Vom kleinsten intelligenten Sensor, über Operator-Bedienpanels wie beispielsweise Supervisory Control and Data Acquisition, über die MES-ERP-Ebene bis hin zu Geräten auf Konsumentenseite, wie Tablets oder Smartphones. [13] Für die Forschungsfrage dient UA somit als Basis der standardisierten Kommunikation, mit der sich im weiteren Verlauf spezifische Daten eines Gerätes unabhängig des Herstellers und unabhängig der eigentlichen Information abrufen, visualisieren und modifizieren lassen. Die Gründe zur Nutzung von UA als Basis zur Geräteintegration sind in der Abgrenzung dieser Arbeit in Kapitel 1 erläutert worden. In Analogie zur UA Technologie soll für die Repräsentation etablierter Kommunikations-

technologien zum Nachweis der Forschungsfrage, FDT als solche Geräteintegrationstechnologie verwendet werden.

3. Analyse

Gemäß der Forschungsfrage gilt es einen Nachweis in Form einer prototypischen Softwareanwendung zu erbringen, der die Frage beantwortet, wie sich verschiedene, spezifische Daten eines Gerätes unabhängig von dessen Hersteller und unabhängig von der eigentlichen Information unter Berücksichtigung etablierter Geräteintegrationsstandards unabhängig des Betriebssystems und des Endgerätes selbst, abrufen, visualisieren und modifizieren lassen. Grundlage zur Beantwortung dieser Frage sind die in Kapitel 2.2 erwähnten Technologien UA und FDT.

In jedem Softwareprojekt stellt die Analyse die Grundvoraussetzung für eine erfolgreiche Realisierung dar. Bei der Lösungsfindung zur Ausgangsproblematik, die im Anschluss an die Analyse in Kapitel 4 schließt, konkretisiert der Fokus auf Auswahl passender in den folgenden Kapiteln beschriebenen Technologien das Fundament der zur entwickelnden Anwendung. [33, 34] Entgegengesetzt der traditionellen Analyse stehen mit UA und FDT die Technologien zur in der Problemstellung definierten Zielerreichung bereits fest. Der Gegenstand der Analyse fußt somit auf der Betrachtung der Möglichkeiten, die sich durch die gewählten Technologien ergeben.

3.1. Technologien

Bei der folgenden Betrachtung handelt es sich um eine reine formale Bestandsaufnahme der Möglichkeiten, die sich mit den gewählten Technologien realisieren lassen. Ziel hierbei ist die Erforschung dieser technologischen

Chancen zur Zielerreichung der in der Forschungsfrage definierten Anforderung zum Abruf, zum Visualisieren und zum Modifizieren von Daten. Technische Anforderungen für die zu entwickelnde Anwendung, welche nicht eindeutig aus den Spezifikationsdokumenten der Technologien hervorgehen, werden zu diesem Zeitpunkt nicht definiert. Dies gewährleistet einen neutralen und unabhängigen Blick auf die Technologie, bei denen ansonsten möglicherweise bereits Wertungen eingeflossen wären. Insbesondere wird damit die unbewusste Manipulation der technologischen Anforderungen verhindert, die im Ergebnis dazu führen können, dass eine Realisierung nicht durchführbar ist.

3.1.1. OPC Unified Architecture

Mit der Technologie UA können Daten zwischen Anwendungen mittels des Softwarearchitekturprinzips Client/Server ausgetauscht werden. Eine Anwendung innerhalb des UA Technologiekontexts kann dabei ein UA Client oder ein UA Server sein. In Relation zum erwähnten Architekturprinzip ist ein UA Server eine Anwendung, die als Dienstleister auftritt und den Informationskonsumenten, die entsprechenden Informationen bereitstellt. Ein solcher Informationskonsument wird durch den UA Client repräsentiert, der vom UA Server die entsprechenden von diesem bereitgestellten Informationen konsumiert. [31] Der Informationsaustausch der Anwendungen im beschriebenen Kontext und damit insbesondere der Datenaustausch mit den dahinterstehenden physikalischen Geräten werden dabei durch die im UA Standard verankerten Protokolle und Dienste geregelt. [13] Dienste sind dabei als Einheiten zusammengehörender Funktionen zu verstehen, die eine Steuerung der Informationsverwaltung gewährleisten. Durch sogenannte Dienstschnittstellen kann auf definierte Dienste zugegriffen werden. [35, 36, 37]

Betrachtet man die einzelnen von UA angebotenen Dienste, lassen sich daraus ganz generische Anwendungsfälle extrahieren, welche die Möglichkeiten der Technologie beschreiben. Führt man diesen generischen Anwendungsfällen einen Detaillierungsgrad hinzu, erhält man Anforderungen, die als Programmlogik zu realisieren sind. Tabelle 3.1 ordnet UA Dienste beziehungsweise die

Orchestrierung von UA Dienste zu den daraus extrahierten generalisierten Anwendungsfällen.

Tabelle 3.1.: Zuordnung von Open Platform Communications Unified Architecture Diensten beziehungsweise Dienstorchestrierung, zu den daraus extrahierten Anwendungsfällen [36]

Dienst / Dienstorchestrierung	**Anwendungsfall**
Discovery Services Set	Server finden
Secure Channel Service Set, Session Service Set	Verbindungsverwaltung
View Service Set	Informationen im Adressraum finden
Read and Write Service	Lesen und Schreiben von Daten und Metadaten
Subscription Service Set, Monitored Item Service Set	Datenänderungen und Events abonnieren
Call Service	Server definierte Methoden aufrufen
HistoryRead and HistoryUpdate Service	Verlauf von Daten und Events
Query Service Set	Informationen im komplexen Adressraum finden
Node Management Service Set	Strukturmodifizierung des Serveradressraums

Um über die Relevanz der extrahierten Anwendungsfälle für die Zieldefinition der Forschungsfrage zu entscheiden, werden gemäß der Erläuterung aus Kapitel 3.1, die generischen Anwendungsfälle in den folgenden Absätzen beschrieben. Eine Gesamtauflistung mit Beschreibung der generischen und den daraus durch den Detaillierungsgrad resultierenden spezifischen Anwendungs-

fällen findet sich in Anhang B.1.

Verbindungsverwaltung Eines der wichtigsten Kriterien für den Erfolg von Industrie 4.0 ist der Transport und die Sicherheit aller im Kontext relevanter Informationen. UA schafft hierfür mit der serviceorientierten Architektur auf oberer Abstraktionsebene und Nutzung etablierter und standardisierter Transportstandards auf unterer Ebene die nötigen Erfordernisse. [13, 36] Die grundsätzlichen Anforderungen die eine sichere, flexible und zuverlässige Datenkommunikation zwischen UA Anwendungen gewährleistet werden in Abbildung 3.1 veranschaulicht. Die Konzepte zur Realisierung der Verbindungsverwaltung sind im UA Standard definiert und werden in der prototypischen Implementierung durch die Portierung des UA Stacks abstrahiert, welche in Kapitel 5.3 beschrieben ist. Eine genauere Definition der durch den Standard aufgeführten Verbindungsverwaltung findet sich im Anhang A.1.2.

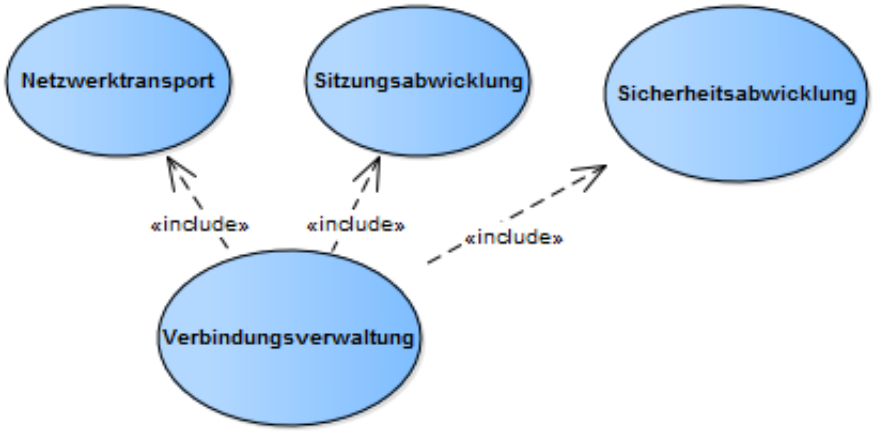

Abbildung 3.1.: Die für den Anwendungsfall - Verbindungsverwaltung - spezifisch ermittelte Anwendungsfälle (basiert auf Informationen aus [36])

Server finden Der Anwendungsfall ermöglicht innerhalb einer administrativen Domäne unter definierten Bedingungen alle verfügbaren UA Server zu identifizieren und entsprechende Endpunkte abzufragen, mit denen eine Sitzung, welche in Anhang A.1.4 als Exkurs gelistet ist, vom Client zum Server

eröffnet werden kann. Die Bedingungen, unter denen ein UA Server innerhalb der administrativen Domäne gefunden werden kann, ergeben sich durch die technischen Begrenzungen der eingesetzten Protokolle. Um Möglichkeiten zur Erfüllung dieser Bedingungen zu gewährleistet, definiert UA in dessen Standard für die verschiedenen Aggregationen der entsprechenden Bedingungen sogenannte Konzepte des *Discovery* (siehe auch Anhang A.1.3). [36, 37, 38, 39] Bei der Implementierung in Kapitel 5.4 werden die einzelnen Implementierungskomponenten, welche nicht durch den UA Stack abstrahiert wurden, im Detail erläutert. Die in der Abbildung 3.2 spezifischen Anwendungsfälle zeigen die verschiedenen Konzepte des Discovery.

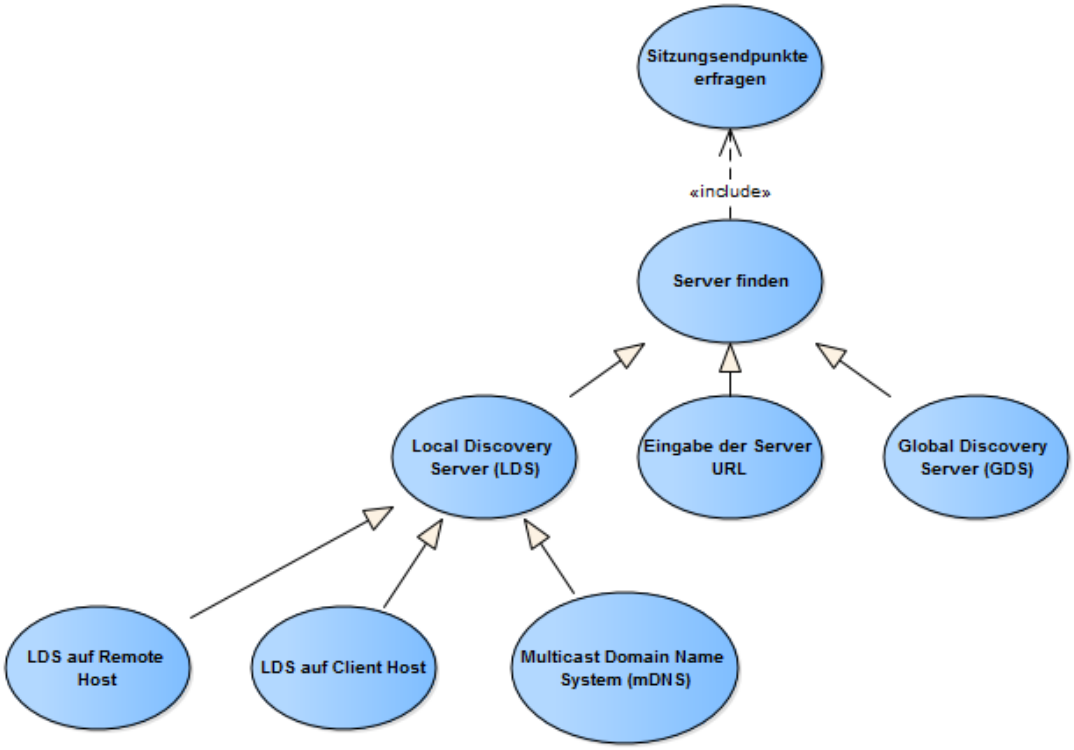

Abbildung 3.2.: Die für den Anwendungsfall - Server finden - spezifisch ermittelte Anwendungsfälle (basiert auf Informationen aus [36, 37, 38, 39])

Informationen im Adressraum finden Bei UA werden zusammenhängende Informationen in *Datenknoten* gruppiert. Die Gesamtmenge der Datenkno-

ten bildet ein voll vermaschtes Informationsnetzwerk. Der Teil an Datenknoten, den ein einzelner UA Server als ein solches Informationsnetzwerk bereitstellen kann, wird *Adressraum* des Servers genannt. [36, 40] Die in Abbildung 3.3 ermittelten Anwendungsfälle dienen zum Auffinden verschiedener Typen von Daten in und von einem UA Server bereitgestellten Adressraum.

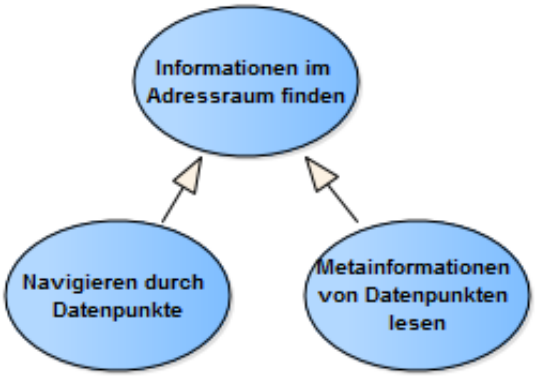

Abbildung 3.3.: Die für den Anwendungsfall - Informationen im Adressraum finden - spezifisch ermittelte Anwendungsfälle (basiert auf Informationen aus [36])

Informationen im komplexen Adressraum finden Der Adressraum eines UA Servers kann an Datenknoten reichhaltig sein. Zu diesem Zwecke sollen mit diesem Anwendungsfall, die im vorangegangenen Abschnitt beschriebenen Anwendungsfälle erweitert werden, die ein grundlegendes Finden von Informationen im Adressraum ermöglichen. Der UA Standard definiert hier einen Anwendungsfall, der durch Filterung des Adressraums, Teilmengen des selbigen aufstellt und so zur besseren Auffindbarkeit des gesuchten Datenknotens beiträgt. [36] Eine Detaillierung dieses Anwendungsfalls zur Umsetzung der Programmlogik ist nicht notwendig.

Lesen und Schreiben von Daten und Metadaten Die Struktur von Datenknoten im Rahmen von UA ähneln Objekten aus dem objektorientierten Programmierparadigma. Wie in der objektorientierten Programmierung werden die eigentlichen Dateninhalte der Datenknoten durch Attribute beschrieben.

Folglich kann zwischen spezifischen Daten eines Datenknotens und Metadaten welche die Dateninhalte beschreiben unterschieden werden. [31, 36, 40] Die in Abbildung 3.4 ermittelten Anwendungsfälle decken so gleichermaßen den Abruf von Daten sowie das Modifizieren von Daten als auch von Metadaten ab, welche sich innerhalb der gruppierten Datenknoten befinden. Hierbei wird beim Lesevorgang ein azyklisches Leseverhalten beschrieben.

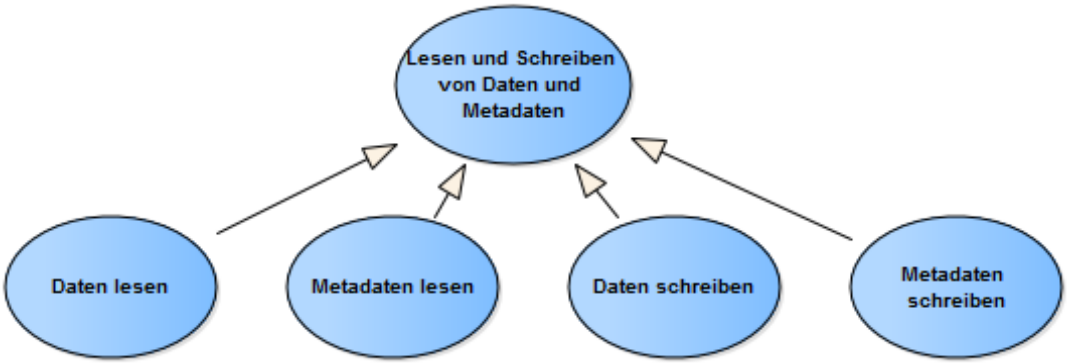

Abbildung 3.4.: Die für den Anwendungsfall - Lesen und Schreiben von Daten und Metadaten - spezifisch ermittelte Anwendungsfälle (basiert auf Informationen aus [36])

Datenänderungen und Ereignisse abonnieren Die gruppierten Daten innerhalb der Datenknoten eines Adressraums können sich abhängig von der Domäne, in der die Daten erhoben und in das Informationsmodell eingespeist werden, in unbestimmten Zeitabschnitten verändern. Damit diese Änderungen vom Informationskonsumenten möglichst direkt nach Auftreten der Wertänderung registriert werden, definiert UA das Konzept des sogenannten Abonnements. Hierzu markiert der Anwender in der Anwendung die zu beobachtenden Datenknoten digital. Durch die digitale Markierung wird der UA Server aufgefordert, den Konsumenten über entsprechende Wertänderungen des markierten Datenknotens zu informieren. Neben Wertänderungen kann sich ein Konsument auch über bestimmte eintretende Ereignisse innerhalb des UA Informationsmodell informieren lassen. [32, 36] In Abbildung 3.5 sind die hierfür nötigen Anwendungsfälle illustriert.

Abbildung 3.5.: Die für den Anwendungsfall - Datenänderungen und Ereignisse abonnieren - spezifisch ermittelte Anwendungsfälle (basiert auf Informationen aus [36])

Server definierte Methoden aufrufen In einem der vorherigen Abschnitte wurde der Vergleich von Datenknoten und Objekten aus dem objektorientierten Programmierparadigma angeführt. Neben dem Einsatz von Variablen und Attributen durch Daten und Metadaten können so auch in einem Datenknoten, Methoden enthalten sein. Genau wie in der objektorientierten Programmierung sind Methoden eine Sammlung ausführbaren Quellcodes, die vom Informationskonsumenten aufgerufen werden können und in aller Regel eine Rückmeldung in Form eines Wertes liefern. [32, 36] Die zur Erfüllung der textuellen Beschreibung notwendigen Anwendungsfälle definiert die Abbildung 3.6 grafisch.

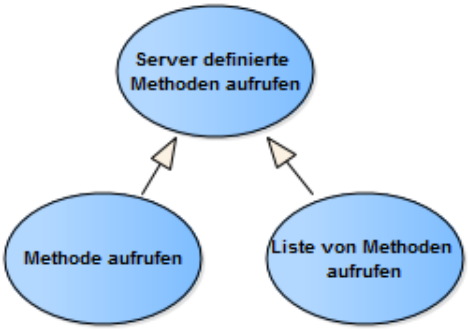

Abbildung 3.6.: Die für den Anwendungsfall - Server definierte Methoden aufrufen - spezifisch ermittelte Anwendungsfälle (basiert auf Informationen aus [36])

Zugriff auf Verlauf von Daten und Ereignisse Ein Verlauf definiert den Zustand eines Datenknotens in einem bestimmten Zeitabschnitt. Mit diesem Anwendungsfall soll die Verwaltung des Verlaufs ermöglicht werden. [36] Eine spezifischere Definition des Anwendungsfalls zur Umsetzung in der Programmlogik ist an dieser Stelle nicht notwendig.

Strukturmodifizierung des Serveradressraums Dieser Anwendungsfall ermöglicht, die Struktur des Serveradressraums zu verändern. Es können exemplarisch Datenknoten hinzugefügt oder entfernt werden. Abbildung 3.7 zeigt die für diesen Anwendungsfall spezifisch ermittelten Anwendungsfälle. [36]

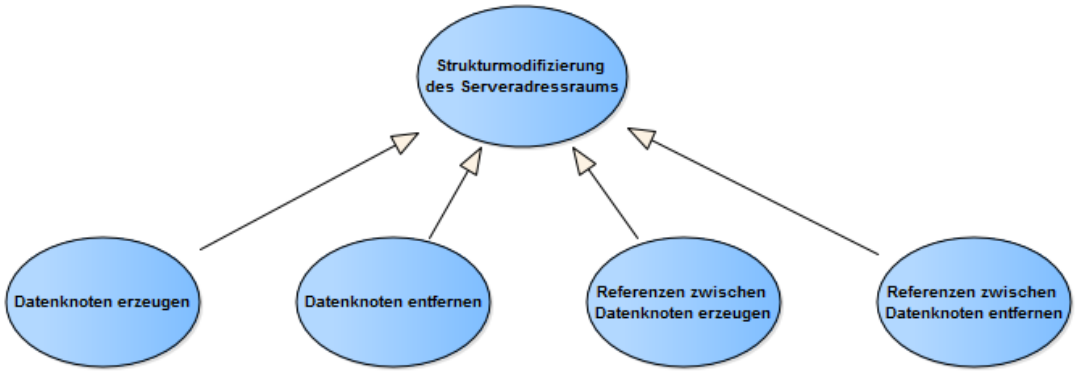

Abbildung 3.7.: Die für den Anwendungsfall - Strukturmodifizierung des Serveradressraums - spezifisch ermittelte Anwendungsfälle (basiert auf Informationen aus [36])

3.1.2. OPC UA Informationsmodell für FDT

Ein wichtiger Aspekt, der im Rahmen dieser Arbeit berücksichtigt werden soll und aus selbigem Grund in der Forschungsfrage definiert wurde, ist die Frage wie sich etablierte Geräteintegrationsstandards im Kontext des definierten unabhängigen Informationsabrufs und in Analogie der unabhängigen Informationsmodifizierung durch UA integrieren lassen.

Bei Betrachtung der industriellen Produktionsdomäne wird die Signifikanz der Integration etablierter Geräteintegrationsstandards für die erfolgreiche Unter-

nehmenstransformation durch Industrie 4.0 deutlich. In der dritten industriellen Revolution, welche durch den Einsatz von Informationstechnologie geprägt ist, haben Unternehmen in den letzten Jahrzehnten massiv in Soft- und Hardwarekomponenten, und viel bedeutender, in die Entwicklung von Geräteintegrationstechnologien auf horizontaler Ebene der Automatisierungs- pyramide investiert. Zur Lösung der in Kapitel 2 erläuterten Kommunikationsproblematik einer Totalvernetzung vom Unternehmen bis hin zu den Produktkonsumenten, wird in selbigem Kapitel die UA Technologie als eine mögliche Lösung vorgestellt. Ein Einsatz neuer Technologien, wie UA führt immer auch unweigerlich zur Substitution von Soft- und Hardwarekomponenten oder aber zu mindestens einer Aufrüstung dieser durch Adaptionslösungen zur Nutzung der neu eingeführten Technologie. Beide Varianten verursachen allerdings massive Eingriffe in den Produktionsablauf und daraus resultierende Stillstandskosten der einzelnen Produktionsstellen. Erschwerend hinzu kommt, dass in der Vergangenheit getätigte Investitionen in Infrastruktur durch eine Substitution dieser verloren gehen. [13, 15, 25]

Zur Vermeidung der Substitution von Produktionsinfrastrukturen sowie der Vermeidung der lokal gebundenen Adaption von Geräten verbleibt als mögliche Variante ausschließlich die zentrale Anpassung der Kommunikationsintegrationsebene als solche. Konkret ist die Kommunikationsintegrationsebene die Menge der verwendeten Geräteintegrationstechnologien auf horizontaler Ebene. [13, 15, 25] Der UA Standard bietet die Möglichkeit das als Norm ausgelieferte Informationsmodell zu erweitern und öffnet somit die Integration horizontaler Technologien für Standardisierungsgremien, die sich in der Vergangenheit auf die Standardisierung von Technologien horizontaler Ebene konzentrierten. [13, 31] Als eine der ersten Organisationen bildet die FDT Group gemeinsam mit der OPC Foundation eine Arbeitsgruppe zur Spezifikation eines für FDT zur Abbildung entsprechender Daten, passenden Informationsmodells mit dem Ziel einer einheitlichen Gerätesicht. Aus den Bemühungen resultiert die Spezifikation *OPC UA Informationsmodell für FDT*. [41, 42]

Mit FDT existiert in Analogie zur UA Technologie eine weitere Geräteintegrationstechnologie, die ebenfalls zum Ziel des Informationsaustausches standardisiert wurde, sich aber auf den Datenaustausch zwischen Feldgeräten und

Automatisierungssystemen horizontaler Integrationsebene beschränkt. FDT wird durch die drei Kernkonzepte, der Rahmenanwendung, des Device Type Manager (DTM) und der Schnittstelle beschrieben. Durch die Beschreibung der Rahmenanwendung, die als grafische Benutzeroberfläche zur Konfiguration, Bedienung und Wartung zu betrachten ist, agiert FDT auf einem höheren Abstraktionsniveau als UA. [41, 43, 44] Als DTM wird in FDT eine Softwarekomponente bezeichnet, die das physikalische Feldgerät repräsentiert. [41] Die Rahmenanwendung als grafische Benutzeroberfläche kann als Softwareumgebung und in Synchronität dazu als Laufzeitumgebung für die DTMs betrachtet werden. [41, 45, 46] Die Interaktion zwischen Rahmenanwendung und DTM findet dabei über die in FDT standardisierten Schnittstellen als drittes Konzept statt. [43, 44, 47]

Hauptanwendungsfälle der spezifischen Informationsmodellvariante *OPC UA Informationsmodell für FDT*, das auf dem Standard-Informationsmodell und der Erweiterung des Standards *OPC UA für Geräte* von UA basiert, sind die Gerätekonfiguration und Diagnose. Dennoch ist es mit dem Informationsmodell möglich, auf jede Art von Daten zuzugreifen, die der entsprechende DTM Repräsentant bereitstellt. [41, 42] FDT als eigenständige Technologie, unabhängig von UA, beschreibt in dessen gleichnamigen Standard bedeutend mehr, als die im *OPC UA Informationsmodell für FDT* abgebildete Anwendungsfälle. [46] Grund für die unvollständige Zuordnung von FDT Anwendungsfällen in besagtes UA Informationsmodell, ist die derzeitige noch anhaltende Entwurfsphase des Modells durch die entsprechende Arbeitsgruppe und die damit unvollständige Abbildung. Ein weiterer Grund ist auf die höhere Abstraktionsebene von FDT gegenüber UA zurückzuführen. So beschreibt der FDT Standard nicht ausschließlich den Gerätehersteller unabhängigen Datenaustausch zwischen Feldgerät und Automatisierungssystemen, sondern auch eine einheitliche, grafische Benutzeroberfläche über, die jedes an die Technologie angebundene Feldgerät auf Feldebene unabhängig von Hersteller, Gerätetyp oder Kommunikationsprotokoll des Geräts konfiguriert, bedient und gewartet werden kann. [41, 44]

Abbildung 3.8 zeigt die generalisierten Anwendungsfälle der UA Technologie sowie die als spezifische integrierte Anwendungsfälle des *OPC UA Informa-*

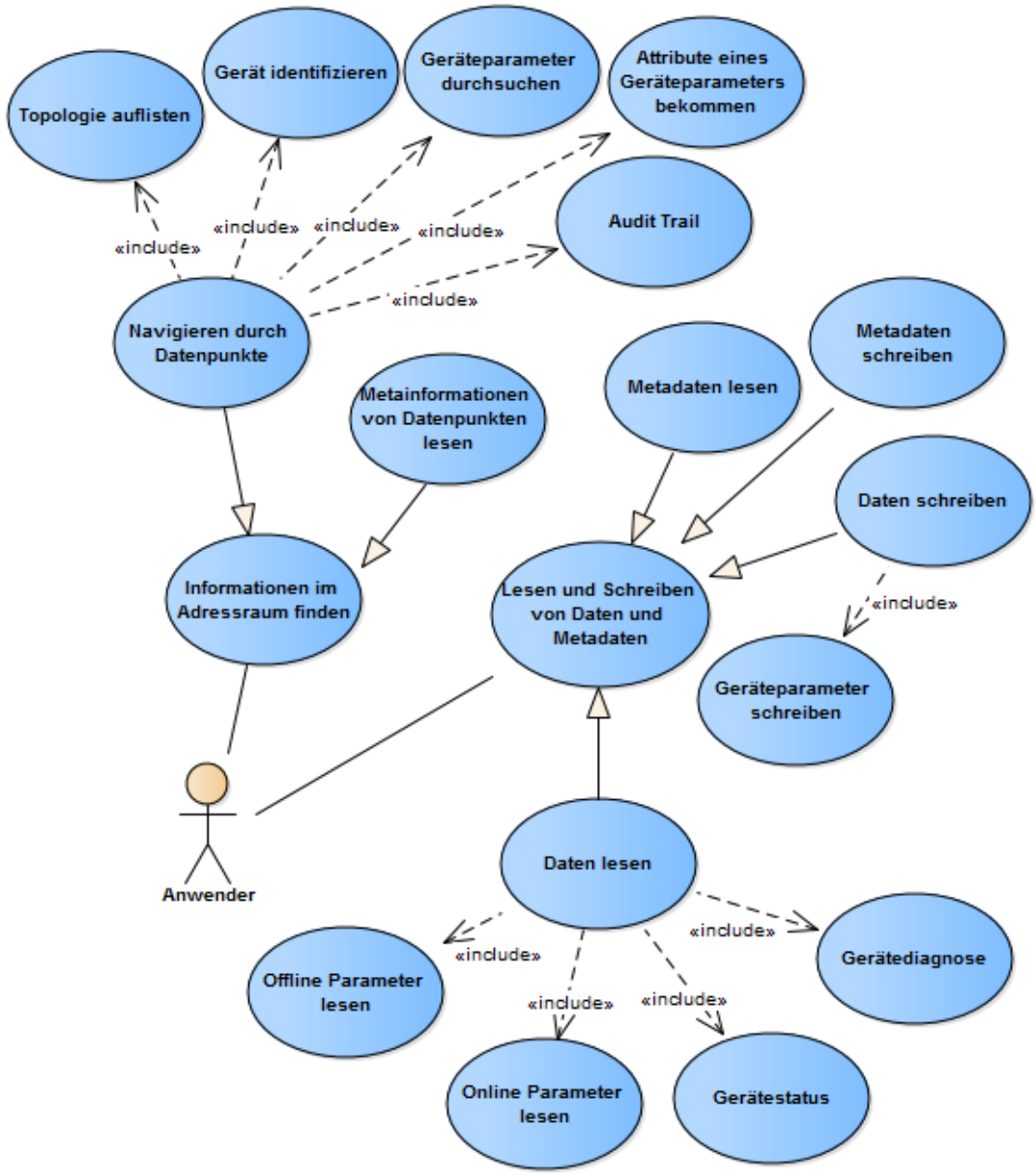

Abbildung 3.8.: Anwendungsfälle des - OPC UA Informationsmodell für FDT - zur Integration und Nutzung von Field Device Tool Anwendungsfällen innerhalb der Open Platform Communications Unified Architecture (basiert auf Informationen aus [41])

tionsmodell für FDT, welche wiederum aus den für das spezifische Informationsmodell abgebildeten Anwendungsfällen der FDT Technologie bestehen. Technisch bedeutet die Integration von Anwendungsfällen des *OPC UA Infor-*

mationsmodell für FDT nach UA, die gleiche Nutzung der Dienste und Dienstaufrufe wie auch beim Standardinformationsmodell. [41] Die durch die UA Technologie über Dienste erreichbare Anwendungsfallmenge kann abstrakt durch $(FDT \cap UA) \cup (UA \setminus FDT)$ beschrieben werden, wobei gilt $FDT \in \{Alle\,FDT2\,Anforderungen\}$ und $UA \in \{Alle\,UA\,Anforderungen\}$. Auf Basis dieser Anforderungsmenge können nun die tatsächlichen Anforderungen für die zu entwickelnde Anwendung definiert werden, zur Erfüllung des durch die Forschungsfrage definierten Problembereichs.

Die nachfolgenden Abschnitte beschreiben die spezifisch in UA integrierten Anwendungsfälle des *OPC UA Informationsmodell für FDT*. Eine Beschreibung aller in der FDT Technologie verfügbaren Anwendungsfälle befindet sich in Anhang B.2 und eine Beschreibung der speziell durch das Informationsmodell *OPC UA Informationsmodell für FDT* abgebildeten Anwendungsfällen auf Grundlage der FDT Anwendungsfälle in Anhang B.3.

Topologie auflisten FDT hat als Geräteintegrationstechnologie in erster Linie zu gewährleisten, dass an ein Gerät adressierte Daten auch beim entsprechenden Gerät eintreffen. Der Weg vom Sender zum Empfänger kann hierbei über unterschiedliche Protokolle und Steuerungsgeräte führen. Die Wege von und zu Geräten selbst werden in FDT durch Verknüpfungen der entsprechenden DTMs repräsentiert, die auch hierarchisch gegliedert werden dürfen. Diese Verknüpfungen werden im FDT Standard als Topologie bezeichnet. [46, 47] Durch Abbildung des Topologiekonzepts im *OPC UA Informationsmodell für FDT* soll ein Datenkonsument die Fähigkeit erhalten, die entsprechende möglicherweise hierarchische Topologie als Baum darzustellen. [41]

Gerät identifizieren Die Realisierung dieses Anwendungsfalls soll einen Datenkonsumenten befähigen, Informationen von in der FDT Topologie eingebundenen physikalischen Geräten abzurufen. Bei den Informationen handelt es sich um die im FDT Standard beschrieben Identifikationsdaten eines Gerätes, welche entsprechend im Informationsmodell *OPC UA Informationsmodell für FDT* abgebildet wurden. [41]

Geräteparameter durchsuchen In FDT wird ein Werthalter eines physikalischen Gerätes auch als Parameter beschrieben. Parameter von Geräten, welche über die FDT Technologie kommunizieren, sind im spezifischen UA Informationsmodell entsprechend abgebildet. Durch diesen Anwendungsfall soll die Anzeige von Parameternamen realisiert werden. Dabei können Parameter in Blöcken gruppiert werden. Ist dies der Fall, sollen auch die Namen der jeweiligen Blöcke angezeigt werden können. [41]

Attribute eines Geräteparameters bekommen Ist ein Parameter als lesbar markiert, soll der Anwendungsfall die Anzeige des Parameterwertes gewährleisten. [41]

Gerätestatus In FDT kann ein physikalisches Gerät verschiedene Status annehmen. Mit diesem Anwendungsfall soll der Status, den ein Gerät angenommen hat, repräsentiert werden können. [41]

Gerätediagnose Die verschiedenen Diagnoseinformationen eines Geräts die FDT abbilden kann, sind ebenfalls im *OPC UA Informationsmodell für FDT* abgebildet worden. Mit diesem Anwendungsfall soll der Zugriff auf diese Informationen ermöglicht werden. [41]

In FDT werden zwischen Offline und Online Daten unterschieden. [46, 48, 49] Offline Daten sind dabei die Werte der Geräteparameter, welche in der entsprechenden DTM Instanz gespeichert sind. Online Daten hingegen werden direkt vom physikalischen Gerät über den DTM bezogen. [46, 49] Sowohl die offline als auch die online Geräteparameterdatenstruktur wird jeweils im spezifischen Informationsmodell abgebildet. [41]

Offline Parameter lesen Dieser Anwendungsfall soll den Datenkonsumenten befähigen, auf die zuletzt im DTM gespeicherten Werte eines Geräteparameters zuzugreifen, der durch ein DTM referenziert wird. [41]

Online Parameter lesen Analog zu den offline Daten ermöglicht dieser Anwendungsfall den über den DTM realisierten Zugriff auf die tatsächlich momentan vom physikalischen Gerät repräsentierten Geräteparameterwerte. [41]

Geräteparameter schreiben Dieser Anwendungsfall ermöglicht die Parameterwerte eines Gerätes, die über den DTM referenziert werden, zu konfigurieren beziehungsweise zu modifizieren. [41]

Audit Trail Der Audit Trail in FDT ermöglicht die Aufzeichnungen von Anwendern und deren Änderungen an einem physikalischen Gerät innerhalb einer bestimmten Zeitspanne. Dabei sendet ein DTM im traditionellen Kontext von FDT immer dann eine Audit-Trail-Benachrichtigung an die Rahmenanwendung, sobald Änderungen am physikalischen Gerät vorgenommen werden, welches durch den DTM repräsentiert wird. Der Anwendungsfall ermöglicht solche Audit Trails durch ein zentralisiertes Audit Trail System zu abonnieren. [41, 46]

3.2. Systemkontext

In den vergangenen Analysekapiteln wurde durch Aufzeigen von Anwendungsfällen der Technologien erläutert, dass die Beantwortung der Forschungsfrage grundsätzlich möglich ist. Konkret wurde hierbei gezeigt, dass mit der standardisierten Kommunikationstechnologie UA auf spezifische Daten eines Gerätes unabhängig vom Hersteller des Gerätes und unabhängig der eigentlichen Information zugegriffen werden kann. Die Daten werden dabei aus dem standardmäßigen Informationsmodell von UA konsumiert, das zur Laufzeit den Adressraum eines UA Servers darstellt. Weiterhin wurde im Kapitel 3.1.2 die Möglichkeit aufgezeigt, wie sich Gerätedaten die über die Geräteintegrationstechnologie FDT kommunizieren, ebenfalls in UA integrieren lassen, sodass diese Daten zur Laufzeit ebenfalls von einer Client Anwendung über einen UA Server konsumiert werden können. Hierzu haben die Standardisierungsgremien OPC Foundation und die FDT Group ein gemeinsames spezifisches

Informationsmodell mit der Bezeichnung *OPC UA Informationsmodell für FDT* etabliert, dass eine Erweiterung zum standardmäßig von UA ausgelieferten Informationsmodell bietet.

Eine Beantwortung der Forschungsfrage soll durch den Nachweis von Abruf und Modifizierung der Daten aus beiden Informationsmodellen durch eine prototypische Implementierung erfolgen. Es ist ein Nachweis durch beide Informationsmodelle erforderlich, da mit dem spezifischen *OPC UA Informationsmodell für FDT* in der Theorie die vollständige Beweisführung durchführbar wäre, diese Beweisführung aber nicht praktisch realisierbar ist. Grund hierfür ist die derzeitige anhaltende Entwurfsphase des *OPC UA Informationsmodell für FDT* und die unvollständige Implementierung des Beispielservers zur Auslieferung des spezifischen Informationsmodells. Hierdurch können nicht alle Minimalanforderungen durch das spezifische *OPC UA Informationsmodell für FDT* für die Beweisführung abgedeckt werden, weshalb diese unterstützend durch das Standardinformationsmodell realisiert werden sollen. Die vom Endgerät und Betriebssystem unabhängige, prototypische Implementierung soll hierbei die aus dem Informationsmodell gelesenen Daten zusätzlich visualisieren und über die grafische Benutzeroberfläche modifizieren können. Im weitesten Sinne handelt es sich damit um eine Client Anwendung in Anlehnung an das Client-Server-Architekturprinzip von UA.

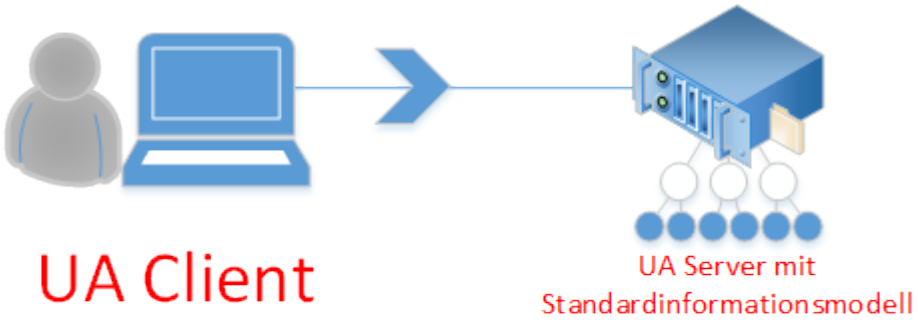

Abbildung 3.9.: Systemkontextdiagramm zur Abbildung des Ausschnitts eines Open Platform Communications Unified Architecture Server, welcher das in der Technologie standardisierte Standardinformationsmodell integriert

Für den Systemkontext bedeuten zwei Arten von Informationsmodellen, dass

die prototypische Client Anwendung mit zweierlei Arten von UA Servern kommunizieren muss. Zum einen mit einem UA Server, welcher regulär das Standardinformationsmodell ausliefert, wie Abbildung 3.9 zeigt, zum anderen mit einem UA Server der die Gerätedaten, die durch FDT bereitgestellt werden in das spezielle *OPC UA Informationsmodell für FDT* abbilden, wie Abbildung 3.10 zeigt.

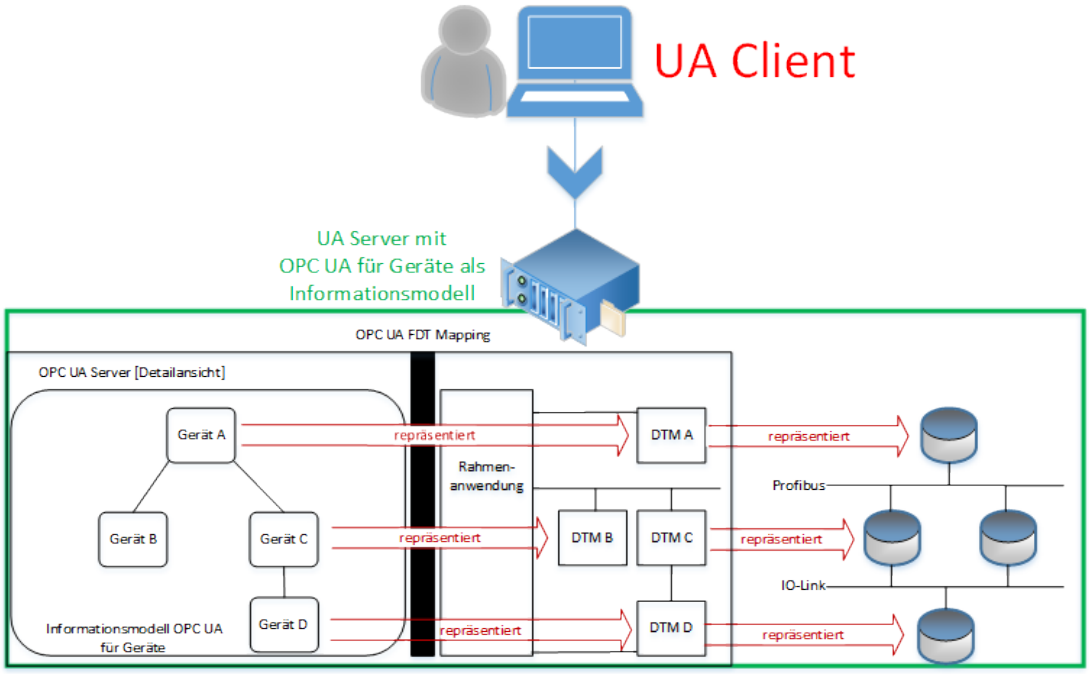

Abbildung 3.10.: Systemkontextdiagramm zur Abbildung des Ausschnitts eines Open Platform Communications Unified Architecture Server, welcher das spezifische OPC UA Informationsmodell für FDT integriert

Die in Kapitel 3.1.1 erwähnen Discovery Konzepte um einen Server innerhalb einer administrativen Domäne aufzufinden erweitern den Systemkontext zusätzlich. Der Grund hierfür ist, dass einige der Discovery Konzepte zur Realisierung des jeweiligen Konzepts, sogenannte Discovery Server einsetzen. Die Aufgabe eines Discovery Server ist die Verwaltung einer Liste, die alle innerhalb eines zusammenhängenden Netzwerkes verfügbare UA Server enthält. Discovery Server können in globale und lokale Server kategorisiert werden. Ein globaler Discovery Server hat dabei die Eigenschaft, dass dieser innerhalb der gesamten administrativen Domäne erreichbar ist. Ein lokaler Discovery

Server hingegen teilt sich in aller Regel eine Systemumgebung im Netzwerk mit den UA Anwendungen, die der Server in seiner Liste aufnimmt. Wenn der lokale Discovery Server auf einer eigenständigen Systemumgebung im Netzwerk existiert, ist sein Wirkungsbereich durch die Grenzen des Subnetzwerkes, in dessen er vorkommt begrenzt. [38] Abbildung 3.11 zeigt den gesamten Systemkontext mit exemplarischen UA Servern, welche die unterschiedlichen Informationsmodelle verwenden. Dabei wird der Netzwerkkontext durch Aufzeigen von jeweiligen Discovery Server in unterschiedlichen Subnetzwerken aufgezeigt.

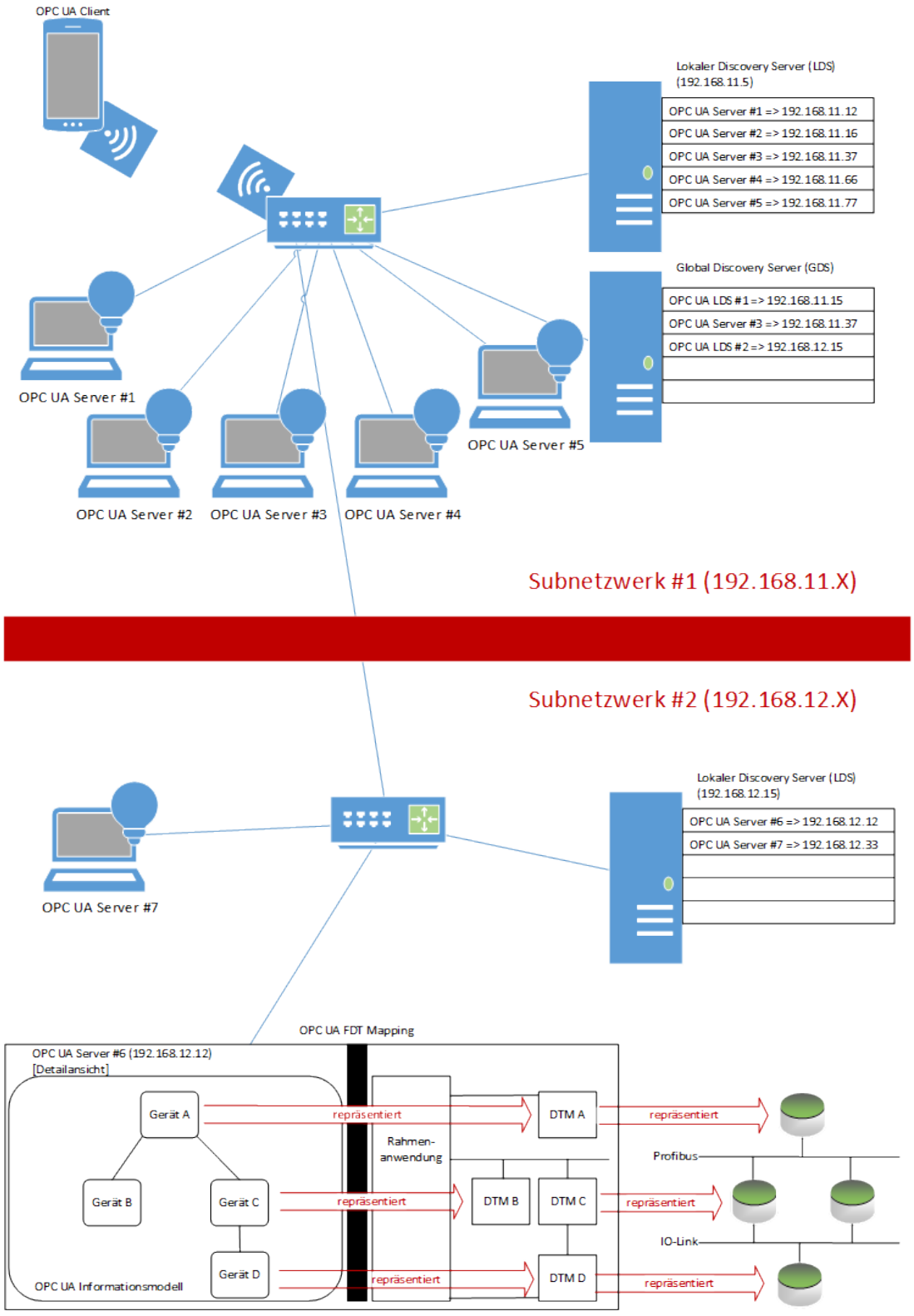

Abbildung 3.11.: Gesamtsystemkontext

3.3. Anforderungen

Im vergangenen Kapitel 3.1 wurden die Möglichkeiten der zu verwendenden Technologien gemäß der Forschungsfrage aufgeführt und in Hinblick auf Aggregation, der beider Technologien analysiert. Daraus resultierte eine Menge von Anforderungen, die sowohl Geräte unterstützen, welche direkt über UA kommunizieren können als auch über die hier exemplarisch verwendete in UA integrierte Technologie FDT.

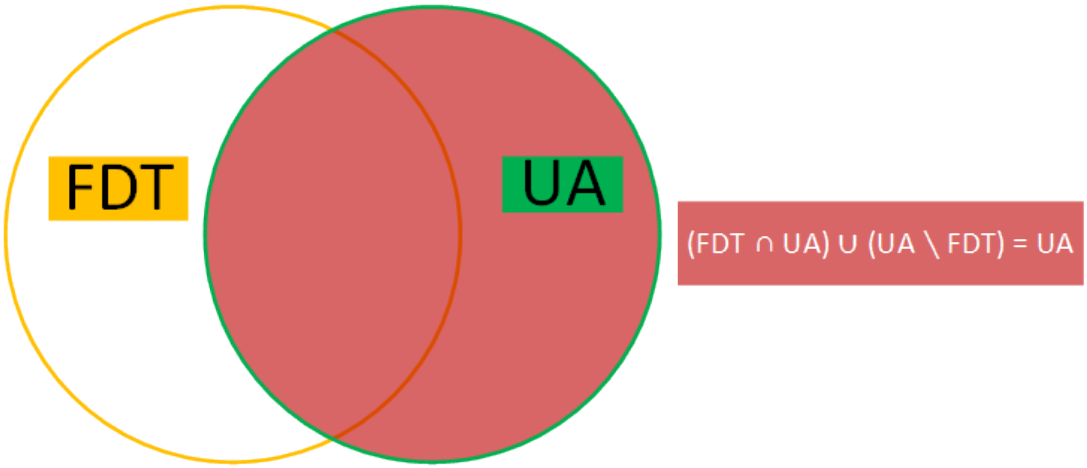

Abbildung 3.12.: Abstrakte Menge, der von Open Platform Communications Unified Architecture und Field Device Tool gemeinsam unterstützten Anforderungen die durch Abbildung vollständig auf die generalisierten Open Platform Communications Unified Architecture Anforderungen abgebildet werden können

Abbildung 3.12 zeigt abstrakt die zu unterstützende Menge an Anforderungen, welche in Abbildung 3.8 aus Kapitel 3.1.2 bereits illustriert wurden. Mit der Abbildung wird deutlich, dass eine Teilmenge von FDT Anforderungen auf die generalisierten Anforderungen von UA durch das *OPC UA Informationsmodell für FDT* abgebildet werden können, welche durch die Schnittmenge illustriert wird und technisch über die Dienste der UA Technologie zur Ausführung gebracht werden können. Mit dieser Menge an Anforderungen können nun die konkreten Anforderungen für die zu entwickelnde Anwendung erhoben werden. Dabei werden die Minimalanforderungen erhoben, die zur Beantwortung des Ausschnitts 'Abruf der Daten' der Forschungsfrage notwendig sind. Dies

bedeutet, dass auch Anforderungen erhoben werden, die zwar nicht direkt zur Beantwortung des Ausschnitts der Forschungsfrage führen, die aber zwingend benötigt werden, um überhaupt einen Datenzugriff unabhängig des UA Informationsmodells zu ermöglichen.

3.3.1. Finden und Verbinden

Bevor Daten von einer UA Anwendung gelesen werden können, besteht das Erfordernis solche UA Anwendungen innerhalb der administrativen Domäne zu identifizieren und sich im Anschluss mit dieser Anwendung zu verbinden. Der UA Standard [38] verwendet hierzu die Begrifflichkeit 'Discovery'. Der Standard beschreibt dabei mehrere Konzepte des Discovery, deren Existenz auf die flexible unabhängige Wahl des Standortes der UA Anwendungen innerhalb der administrativen Domäne und die Kommunikation basierend auf TCP zurückzuführen sind. Die Konzepte können in zwei Kategorien unterscheiden werden. Bei der ersten Kategorie handelt es sich um Konzepte, bei denen die Adressen zur Erreichung des UA Servers unmittelbar durch explizites Wissen des Anwenders oder durch Hinterlegung in der Abrufanwendung bekannt sind. Bei der zweiten Kategorie ist der UA Server nicht explizit bekannt, sondern muss durch Erfragung anderer UA Teilnehmer identifiziert werden. Bei Unterstützung mindestens eines Konzepts aus den jeweiligen Kategorien ist der Nachweis dieses Anwendungsfalls in Abhängigkeit zur Forschungsfrage nachgewiesen.

3.3.2. Adressraum durchsuchen

Nachdem eine Verbindung zum UA Server durch Erfüllung der in Kapitel 3.3.1 beschriebenen Anforderung hergestellt wurde, müssen die gewünschten abzurufenden Daten im Adressraum gefunden werden können. Dies ist notwendig, um die in der Forschungsfrage definierten 'spezifischen Daten' eines Gerätes im Anschluss daran zu lesen, zu visualisieren und zu modifizieren. Um gewünschte Daten zu finden, muss der Anwender die Möglichkeit besitzen, durch den Adressraum der vom UA Server bereitgestellten Daten navigieren

zu können. Zur Unterstützung der in der Forschungsfrage definierten etablierten Kommunikationstechnologien und die in dieser Arbeit konkret verwendete Informationsmodelllösung OPC UA Informationsmodell für FDT von FDT, muss das Navigieren sowohl durch das Standardinformationsmodell von UA ermöglicht werden, als auch durch das OPC UA Informationsmodell für FDT. Beim Navigieren durch die Datenknoten im Adressraum sollen die Metainformationen wie beispielsweise Namen dieser Datenknoten direkt ersichtlich werden, durch entsprechende Visualisierung. Die in Kapitel 3.1.1 beschrieben Anwendungsfälle zur Durchsuchung des Adressraumes, lassen es zu, den Adressraum einzuschränken, um schneller an Informationen zu gelangen. Für den Nachweis der Auffindbarkeit von Datenknoten innerhalb des Adressraums zur Realisierung des in der Forschungsfrage definierten Abrufs, Visualisierung und Modifizierung ist dies jedoch nicht erforderlich.

3.3.3. Datenknoten lesen

Sobald ein Datenknoten durch die Erfüllung von in Kapitel 3.3.2 beschrieben Anforderung aufgefunden wurde, soll der Anwender die Möglichkeit besitzen, die Attribute und Variablenwerte des Datenknotens zu lesen. Zur Erbringung des Nachweises, dass auch Datenknoten des OPC UA Informationsmodell für FDT gelesen werden können, sollen exemplarisch Datenknoten mit Online Daten eines physikalischen Gerätes und Offline Instanzdaten eines DTMs unterschieden werden können.

3.3.4. Datenknoten schreiben

Neben dem Lesevorgang von Datenknoten sollen die Datenknoten auch aktualisiert werden können. Dabei muss der Datenknoten in erster Instanz wieder durch Erfüllung der Anforderung aus Kapitel 3.3.2 aufgefunden werden. Anschließend können die entsprechenden zu verändernden Attribute und Variablen aktualisiert werden. In Analogie zum Lesevorgang des Datenknotens, wie in Kapitel 3.3.3 beschrieben, sollen beim OPC UA Informationsmodell für

FDT und beim Standardinformationsmodell, Datenknoten aktualisiert werden können.

3.3.5. Datenknoten abonnieren

korrekt) Daten von industriellen Geräten können sich innerhalb kurzer Zeitintervalle verändern. Zum Nachweis, dass mit der Technologie solche Datenänderungen zeitnah erkannt werden können, soll das im von UA definierte Konzept des Abonnements realisiert werden. Hierzu soll der Anwender die Möglichkeit erhalten, vorher festgelegte Datenknoten auf Datenänderungen zu überwachen.

3.3.6. FDT2 Gerätetopologie anzeigen

Während die meisten FDT Konzepte bestehende Datenknoten des UA Standardinformationsmodells umstrukturieren zur Erreichung einer Abbildung dieser Konzepte als OPC UA Informationsmodell für FDT, beschreibt die in Kapitel 3.1.2 definierte Anforderung zur Anzeige der FDT2 Topologie, eine der wenigen Konzepte, die eine neue Datenknotenhierarchie im Informationsmodell zur Abbildung einbinden. Zur Überprüfung, ob auch diese Informationen mit einer auf UA basierten Anwendung abgerufen werden können, soll die Gerätetopologie von FDT entsprechend dargestellt werden und die Identifikationsdaten der jeweiligen in der Topologie vorhandenen Geräte visualisiert werden.

4. Konzeption und Entwurfsentscheidungen

Das Analyseergebnis aus Kapitel 3 zeigt, dass im Kontext der vierten industriellen Revolution mit der Technologie UA eine standardisierte Kommunikationstechnologie auf theoretischer Basis durch den von UA definierten Standard existiert, mit welcher spezifische Daten eines Gerätes unabhängig des Herstellers, des Gerätes und der eigentlichen Information abgerufen werden können. Weiterhin zeigte das Analysekapitel, dass auch eine theoretische Einbindung der etablierten Geräteintegrationstechnologie FDT in UA durch das *OPC UA Informationsmodell für FDT* integriert werden kann. Die theoretischen Erkenntnisse der Technologien durch Analyse sollen nun durch eine prototypische Softwareimplementierung auch für den praktischen Einsatz, insbesondere unter der in der Forschungsfrage definierten Plattformunabhängigkeit nachgewiesen werden. Die in der Analyse in Kapitel 3.3 definierten Minimalanforderungen ergeben die im Prototyp zu implementierenden Funktionalitäten zum technologischen Nachweis des Datenabrufs, der Datenvisualisierung und der Datenmodifizierung. Die folgenden Unterkapitel zeigen die notwendigen erarbeiteten Konzepte um den Nachweis zu erbringen, der zur Beantwortung der Forschungsfrage führt. Die nachfolgend aufgestellten, analysierten und zur Entwurfsentscheidung aufgestellten Konzepte, bilden mit dem Ergebnis der Entwurfsentscheidung die Grundlage für das in Kapitel 5 beschriebene Entwurfsdesign und die Implementierung der prototypischen Anwendung.

4.1. Entwicklung plattformübergreifender Anwendungen

Eine einheitliche Definition des Verständnisses von Plattform- unabhängigkeit ist im Rahmen dieser Arbeit von besonderer Bedeutung, um in den nachfolgenden Konzepten als Problemfaktor Plattform- unabhängigkeit zu beachten.

Wird in dieser Arbeit von Plattform- unabhängigkeit gesprochen, so ist eine Unabhängigkeit von Betriebssystem und Endgerät gemeint, zur Erreichung der Zieldefinition von Datenabruf, Datenvisualisierung und Datenmodifizierung spezifischer Gerätedaten über UA.

In den letzten Jahren hat die Verbreitung sogenannter smarter Geräte deutlich zugenommen. Exemplarisch für smarte Geräte sind Smartphones, Smartwatches oder Tablets. Softwareentwicklern eröffnet die Verbreitungszunahme ein großes Publikum für zu distribuierende Anwendungen, die zugleich die Entwicklung solcher Anwendungen bei Betrachtung der in der Vergangenheit getätigten Entwicklungsarbeit erschwert. Der Grund für die erschwerte Entwicklung ist auf die Heterogenität der Hardwareeigenschaften von Geräten zurückzuführen, wie beispielsweise unterschiedliche Bildschirmauflösungen oder verschiedene Rechenleistungen. [50] Abgesehen von der Heterogenität der Hardware, ergibt sich mit der unterliegenden Plattform der Anwendung, die in variantenreicher Vielfalt auf dem Markt anzutreffen sind, die größte Problematik. Exemplarisch seien hier die Plattformen Android, iOS und Windows Phone erwähnt, die in Kombination den Markt dominieren. [50, 51, 52, 53]

Die Entwicklung und Wartung von Softwareanwendungen für unterschiedliche Plattformen ist kosten- und zeitintensiv. Aus diesem Grund beschränken sich Entwickler bei der Entwicklung von Anwendungen meist auf eine kleine Anzahl von Plattformen. Die Anzahl an möglich zu erreichenden Anwendern kann hierdurch stark reduziert werden. [50] Um den erwähnten Problematiken gemäß der Forschungsfrage zur Entwicklung von Anwendungen für unterschiedliche Betriebssysteme und Endgeräts entgegenzuwirken, wurden in den vergangenen Jahren verschiedenste Entwicklungswerkzeuge hergestellt. Grundgedanke bei der Entwicklung dieser Werkzeuge war die einmalige von der Plattform unabhängige Entwicklung der Anwendung, welche dann durch die entsprechenden Werkzeuge auf den unterschiedlichsten Plattformen und Endgeräten ausgerollt werden können. Das Resultat der meisten dieser Entwicklungswerkzeuge ist keine native Anwendung für die jeweilige Plattform, sondern eine Web- oder Hybridanwendung. [50, 54]

Native Applikationen sind speziell für die jeweilige Plattform zugeschnitten und

nutzen das Verhalten und Aussehen, wie es Anwender der Plattform gewohnt sind. Native Applikationen erreichen das beste für die Plattform mögliche Zeitverhalten bei der Nutzung der Rechenleistung des jeweiligen Endgeräts. Webanwendung hingegen verwenden standardmäßige Webtechnologien wie beispielsweise Hypertext Markup Language (HTML) 5, JavaScript oder Cascading Style Sheets (CSS). Neben Schwierigkeiten das gewohnte für den Anwender bekannte Verhalten und Aussehen in der Webanwendung zu imitieren, ergibt sich eine besondere Problematik mit dem Zugriff auf native Funktionalitäten, wie beispielsweise die Kamera, des über die Plattform verwalteten Geräts. Hybride Anwendungen sind eine Mischung aus nativen und Webbasierten Anwendungen. Eine durch Webtechnologien entwickelte Anwendungskomponente kann bei einer hybriden Anwendung in einem auf die Plattform abgestimmten nativen *Container* ausgeführt werden. Hierdurch können Schwierigkeiten von webbasierten Anwendungen, wie dem Zugriff auf native Funktionalitäten vereinfacht werden. [55] Problem der Anwendungen, die mit entsprechenden Entwicklerwerkzeugen hergestellt wurden, ist gemäß der beschriebenen Nachteile von Web- und Hybridanwendungen die deutlich schlechtere Nutzungsmöglichkeit von Interaktionselementen und Grundfunktionalitäten. Hinzu kommt, dass unterschiedliche Werkzeuge eine unterschiedliche Menge von Varianten für die Plattformen unterstützen, welches es deutlich erschwert das für die festgelegten Anforderungen passende Entwicklungswerkzeug zu finden. [50, 54]

In Kapitel 1 wurde mit Xamarin das Entwicklungswerkzeug zur Herstellung der prototypischen Anwendung bereits definiert. Im Gegensatz zu anderen Werkzeugen, welche die von der Plattform unabhängige Entwicklung ermöglichen, kann mit Xamarin nativer Quellcode für die jeweilige Plattform ausgehend vom gemeinsamen an die Plattform zu tauschenden Quellcode, generiert werden. [56] Xamarin ermöglicht dies für die Plattformen: iOS, Android und Windows. [56, 57] Hierzu werden die nativen Programmierschnittstellen von iOS und Android, die zur Entwicklung nativer Anwendungen für die jeweiligen Plattformen genutzt werden, von Xamarin auf C# Programmierschnittstellen Aufrufe abgebildet. Xamarin ist somit mehr als nur ein Framework, es ist eine Sammlung von Werkzeugen zur plattformübergreifenden Entwicklung und kann deshalb

als Entwicklungsumgebung bezeichnet werden. Die Entwicklungsumgebung Xamarin basiert dabei auf einer .NET Laufzeitumgebung. [56] Wenn im Folgenden von der prototypischen Anwendung gesprochen wird, so wird hierzu der Anwendungscodename *TheClient* verwendet. Angetrieben von Xamarin soll die prototypische Implementierung von TheClient grundsätzlich alle Plattform unterstützen, welche auch von Xamarin unterstützt werden können. Konkret bedeutet dies die Unterstützung folgender Plattformen: iOS, Android, Windows 8.1, Windows Phone 8.1 sowie Universal Windows Platform (UWP).

4.2. Auswahl Quellcodeverteilungsstrategie

Aus dem vergangenen Kapitel 4.1 geht die Notwendigkeit hervor, zur Erarbeitung eines Konzeptes, welches die Verteilung des einmalig niedergeschriebenen Quellcodes auf die unterschiedlichen Plattformen ermöglicht. Die Entwicklungsplattform Xamarin hat hierzu zwei Strategien zur Quellcodeverteilung erarbeitet: Gemeinsame Projekte [58] und Portable Klassenbibliotheken [58]. Die Auswahl, der für die prototypische Implementierung passende Quellcodeverteilungsstrategie ist eine der primären Entwurfsentscheidungen, auf der zukünftige Konzepte und deren Entscheidungen für den Entwurf basieren. Im Folgenden werden die Strategien grob umschrieben, um eine Entwurfsentscheidung zu treffen.

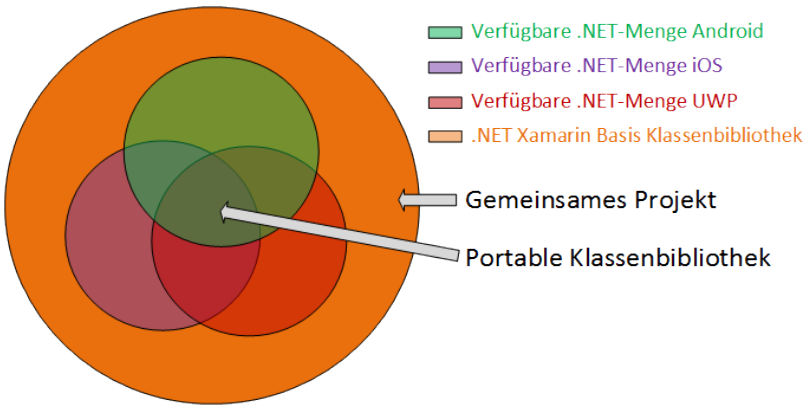

Abbildung 4.1.: Verfügbarkeit der .NET Funktionalitäten bei der Quellcodenutzungsvariante 'Gemeinsames Projekt' und 'Portable Klassenbibliothek'

Die Abbildung 4.1 zeigt die verschiedenen von Xamarin unterstützten und für TheClient relevanten Plattformen. Die unterschiedlichen Plattformen unterstützen nicht die vollständige .NET Basis Klassenbibliothek sondern nur Teilmengen dieser. Hinzu kommt, dass die verschiedenen Plattformen nicht dieselben Teilmengen der .NET Basis Klassenbibliothek unterstützen. [59] Diese Problematik ist ausschlaggebend für die beiden erarbeiteten Quellcodeverteilungsstrategien von Xamarin.

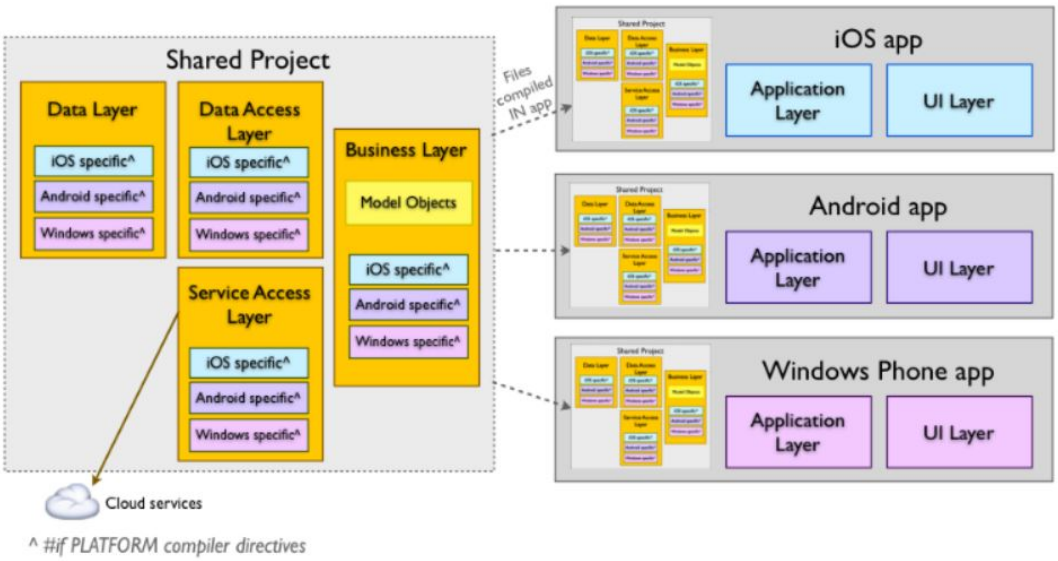

Abbildung 4.2.: Quellcodeverteilung durch ein gemeinsames Projekt [60]

Bei der Quellcodeverteilung über ein gemeinsames Projekt handelt es sich um eine Menge von Quellcodedateien, welche von den verschiedenen plattformspezifischen Projekten wie Abbildung 4.2 illustriert, integriert und kompiliert werden. In diesen Quellcodedateien darf im Quellcode, die vollständige von Xamarin unterstützte Menge der .NET Basis Klassenbibliotheksfunktionalitäten verwendet werden. Der gemeinsame Quellcode, der von allen Plattformen integriert wird, kann nicht eigenständig kompiliert werden. Dieses gemeinsame Projekt wird technisch von den Plattformprojekten referenziert, um so eine Integration zu gewährleisten. [60] Durch Kompilerdirektiven, die exemplarisch in Abbildung 4.3 dargestellt werden, können spezifische Funktionalitäten der Plattform adressiert werden, die nur für diese Plattform gilt. [58, 60] Die in Kompilerdirektiven befindlichen Codeausschnitte werden damit nur für die jeweilige

Plattform kompiliert und in dieser ausgeführt, für welche die Codeausschnitte gelten. [60] Für die Variante des gemeinsamen Projekts lassen sich zwei wesentliche Vorteile ausmachen. Zum einen können Codeausschnitte innerhalb des gemeinsamen Quellcodes auf ein spezifisches Projekt beschränkt werden. Zum anderen besteht beim gemeinsamen Projekt, durch die Unterstützung des vollständig von Xamarin unterstützten .NET Basis Klassenbibliotheksumfang, die Möglichkeit zur Einbindung und Verwendung im gemeinsamen Quellcode von Drittanbieter Softwarekomponenten wie beispielsweise SQLite. Als bedeutsame Nachteile seien die fehlende Möglichkeit eines Ausgabe-Assemblies für das gemeinsame Projekt und das nicht vollständige Refactoring durch fehlende Unterstützung von Refactoring für inaktive Kompilerdirektiven zu erwähnen. Inaktive Kompilerdirektiven sind dabei die Direktiven, die bei Kompilierungsauswahl in der integrierten Entwicklungsumgebung von beispielsweise Android, nicht Android zuzuordnen sind. [58]

```
1  ...
2  #if __IOS__
3          // iOS spezifischer Quellcode
4  #endif
5  ...
6  #if __ANDROID__
7          // Android spezifischer Quellcode
8  #endif
9  ...
10 #if __ANDROID_11__
11         // Quellcode welche nur auf Android Honeycomb (
               ↪ API Version 11) und neuer funktioniert
12 #endif
13 ...
```

Abbildung 4.3.: Verschiedene beispielhafte Kompilerdirektiven im gemeinsamen Quellcode ermöglichen die Implementierung von plattformspezifischen Quellcode

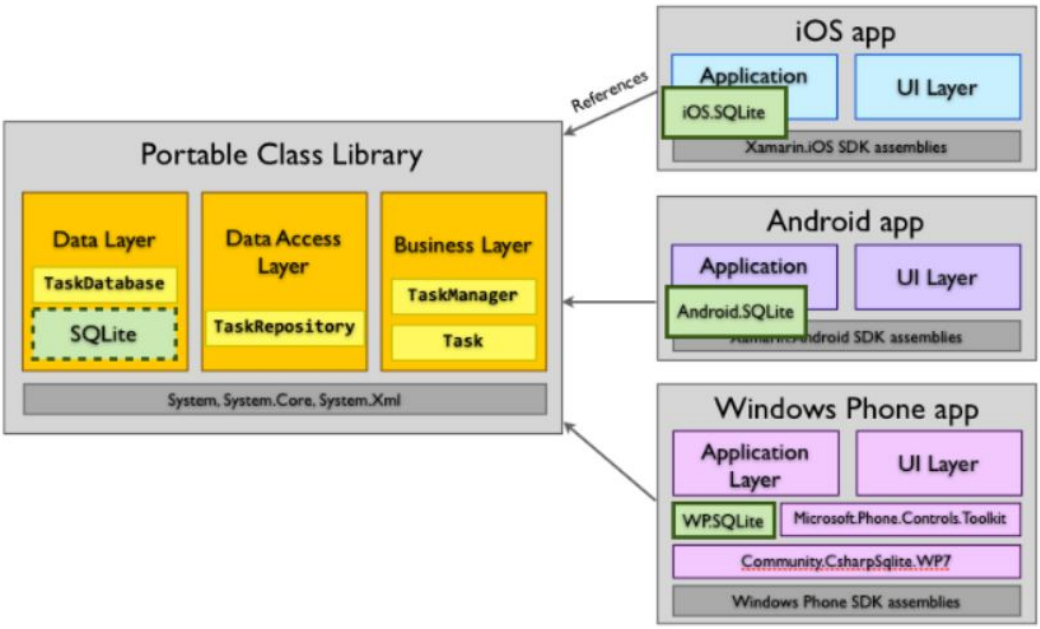

Abbildung 4.4.: Quellcodeverteilung durch eine portable Klassenbibliothek [59]

Bei der zweiten von Xamarin angeboten Quellcodeverteilungsstrategie handelt es sich um portable Klassenbibliotheken. [58] Betrachtet man Applikationsprojekte außerhalb der Entwicklungsdomäne Xamarin, sind die dortigen aus dem kompilierten Quellcode resultierenden DLLs auf in der Regel genau eine spezifische Plattform begrenzt. Die spezifische Plattform ist dabei genau die Plattform, für die der Quellcode geschrieben wurde. Wird beispielsweise eine Anwendung für UWP entwickelt, kann das aus dem UWP-Projekt resultierende Assembly durch Xamarin nicht für Android oder iOS verwendet werden. [59] Abbildung 4.4 zeigt die Verwendung einer portablen Klassenbibliothek in Xamarin die von iOS, Android und UWP referenziert werden kann. Hierbei wird beim Erzeugen der portablen Klassenbibliotheksvorlage in der integrierten Entwicklungsumgebung angegeben, welche Plattformen unterstützt werden sollen. Daraus resultiert ein Entwicklungsprojekt dass ausschließlich Quellcode mit .NET-Referenzen kompilieren kann, wenn diese .NET-Referenzimplementierungen auf allen Plattformen, die bei der Erzeugung der portablen Klassenbibliotheksvorlage angegeben wurde, auch unterstützt

werden. Der Vorteil der portablen Klassenbibliothek gegenüber der Variante des gemeinsamen Projekts ist die Möglichkeit der vollständigen Ablegung des Quellcodes in einem zentralisierten Projekt, welcher dann entsprechend von den Plattformprojekten referenziert wird, ohne dass einzelne Codestellen die auf .NET-Referenzen verweisen in einem plattformspezifischen Projekt abgelegt werden müssen. Dies ermöglicht ebenfalls zentralisiertes Testen des Quellcodes. Da die portable Klassenbibliothek eigenständig kompiliert und als Assembly verteilt und von anderen Anwendungen konsumiert werden kann, ist die Wiederverwendbarkeit von Quellcode in portablen Klassenbibliotheken deutlich höher anzusetzen als in gemeinsamen Projekten. Als weiterer Vorteil ist das Refactoring zu erwähnen, dass bei der portablen Klassenbibliothek den vollständigen Quellcode betrifft und so eine deutliche Zeiteinsparung nach sich ziehen kann. Nachteil dieser Variante ist die fehlende Möglichkeit zur Einbindung von plattformspezifischen Softwarekomponenten von Drittanbieter, welche die .NET-Schnittmenge aller durch die portable Klassenbibliotheksvorlage gewählter Plattformen nicht erfüllen. Eine Unterstützung für Klassen, welche ansonsten in MonoTouch und Mono für Android verfügbar sind, ist ebenfalls nicht gegeben. Durch das Entwurfsmuster des Dependency Injection lassen sich Quellcodeabschnitte, die spezifische Funktionalitäten einer ganz bestimmten Plattform benötigen, in Analogie zu den Kompilerdirektiven des gemeinsamen Projekts, als plattformspezifischer Quellcode kompilieren. Durch die Verwendung dieses Entwurfsmusters lassen sich auch bestimmte Quellcodeabschnitte nutzen, die auf .NET-Referenzimplementierungen verweisen und von der Plattform nativ durch Xamarin unterstützt würden, aber durch die portable Klassenbibliothek aufgrund der Einhaltung der .NET Funktionsschnittmenge aller Plattformen substituiert wurden. [58, 59]

Die folgenden Auswahlkriterien stellen die Auswahlmenge für die Wahl der passenden Quellcodeverteilungsstrategie für TheClient dar:

Drittanbieter Funktionalitäten Es muss davon ausgegangen werden, dass TheClient Funktionalitäten von Drittanbietern konsumieren wird. Dies können zum Beispiel SDKs zur Abstraktion der Kommunikation zwischen UA Client

und UA Server sein oder die Nutzung von Datenbanken für die Ebene der Persistenz.

Quellcode Unabhängigkeit Weiterhin muss davon ausgegangen werden, dass der Quellcode nicht für die Nutzung in plattformübergreifenden Projekten unter Xamarin entwickelt wurde. Das bedeutet, dass der Quellcode gegebenenfalls spezifische Befehle für eine Plattform verwendet oder aber eine nicht unterstützte .NET-Menge nutzt.

Die beiden Auswahlkriterien sind bereits ausreichend zur Entscheidung für eine Quellcodeverteilungsstrategie. Gemäß den oben beschriebenen Vorteilen der beiden durch Xamarin angebotenen Strategien, eignet sich für die prototypische Implementierung von TheClient die Variante des gemeinsamen Projekts bei erster Betrachtung am besten. Die Verwendung von Drittanbieter Funktionalitäten wie beispielsweise SDKs zur Abstraktion von Funktionalitäten in TheClient ist sehr wahrscheinlich. Es ist davon auszugehen, dass diese Drittanbieter Funktionalitäten für alle Plattformen verwendet werden sollen, welches für die zentralisierte Einbindung externer Assemblies durch das gemeinsame Projekt spricht. Weiterhin spricht für das gemeinsame Projekt das im gemeinsamen Quellcode spezifische Anwählen von Plattformen zur Integration plattformspezifischen Quellcodes. Der Nachteil eines fehlenden Ausgabe-Assemlies speziell für das gemeinsame Projekt hat keine Signifikanz für die prototypische Implementierung und kann deshalb ignoriert werden. Auch wenn alles für das gemeinsame Projekt spricht, hat der Vorteil des speziellen Auswählen von plattformspezifischen Projekten über Kompilerdirektiven gleichzeitig den Nachteil der schnellen Unübersichtlichkeit und Lesbarkeit. Problematiken, die bei der portablen Klassenbibliothek nicht bestehen. Die Lösung und damit entwurfsentscheidend für die Quellcodeverteilungsstrategie für TheClient ist die Nutzung der Kombination des gemeinsamen Projekts und der portablen Klassenbibliothek. Hierbei werden alle Vorteile des gemeinsamen Projekts genutzt, die sich aber zugleich des Konzepts bedienen, der Auslagerung von Plattform spezifischen Quellcodes in die Plattform spezifischen Projekten, anstatt über Kompilerdirektiven. Dies soll durch das Entwurfsmuster des Dependency Injection realisiert werden, welches im Implementierungskapitel 5.2

näher beschrieben wird.

4.3. UA Toolkit

Anwendungen, die auf einem solchen komplexen Standard wie dem der UA Technologie basieren, werden nicht von Grund auf neu entwickelt. Durch die Standardisierung von UA ist eine Implementierung der Mechanismen, die zum Abruf und zur Modifizierung von Daten berechtigen, unabhängig der eigentlichen zu entwickelnden Anwendungslogik möglich. Die Implementierung der Mechanismen kann hierbei in der Regel von Drittanbietern konsumiert werden. Hierbei stellen Drittanbietern die Implementierungen der Mechanismen als sogenanntes Software Development Kit (SDK) zum Verkauf. Der UA Standard besteht aus dreizehn Hauptspezifikationen [61] und aus einer noch höheren Anzahl an Kombinationsstandards, welche beispielsweise die Integration etablierender Geräteintegrationsstandards wie FDT innerhalb UA ermöglichen. Bei dieser Komplexität der UA Technologie wird in der Praxis ein Toolkit verwendet. Ein Toolkit wird ebenfalls von Drittanbietern auf Basis des Standards entwickelt und verkauft. Ein Toolkit ist dabei wesentlich komplexer als ein SDK. Im Rahmen von UA kann ein Toolkit in ein SDK und ein sogenannten Stack untergliedert werden. Der Stack übernimmt dabei die Verbindungsverwaltung, den Transport und große Teile der Sicherheitsverwaltung. Das SDK sorgt für die Abstraktion des in der Regel komplexen Stack, um so die Komplexität der Kommunikation zwischen Client und Server Anwendung zu reduzieren. Für die Umsetzung von TheClient ist ein solches Toolkit für den Datenabruf und die Datenmodifizierung zwingend erforderlich, um in einem angemessenen Zeitrahmen den Nachweis zur Beantwortung der Forschungsfrage zu gewährleisten.

4.3.1. Auswahl eines geeigneten UA Toolkits

Auf dem Markt steht eine Anzahl an Toolkits zur Verfügung, die in der Regel unterschiedliche Merkmale aufweisen. Neben Auswahl der passenden Code-

verteilungsstrategie in Kapitel 4.2 ist die Auswahl eines geeigneten UA Toolkits eine zweite primäre Entscheidung, auf deren Grundlage die nachfolgenden Kapitel, Unterkapitel und die Umsetzung von TheClient als solche basieren.

Bei der Auswahl eines für TheClient passenden Toolkits sind diverse Rahmenbedingungen zu beachten. Diese Rahmenbedingungen ergeben sich dabei hauptsächlich durch die Entwicklungsplattform Xamarin, die dadurch wie folgt abgeleitet werden können:

Kompiliertes Toolkit muss vollständig funktional sein oder Toolkit Quellcode muss vorliegen Ein kompiliertes Toolkit muss von der Entwicklungsplattform Xamarin voll funktional eingebunden werden können. Ist dies nicht möglich, so muss das Toolkit entsprechend angepasst werden, um eine Lauffähigkeit zu ermöglichen. Eine Anpassung des Toolkits kann auf zwei Varianten geschehen. Entweder ist das Toolkit parallel der kompilierten Variante auch als Quellcode verfügbar oder das Toolkit kann dekompiliert werden. Die Auswahl für eine der beiden Varianten ist von folgenden Kriterien abhängig:

a) Quellcode des Toolkits ist käuflich oder unentgeltlich erhältlich oder die kompilierte Variante lässt sich vollständig dekompilieren

b) Rechtliche Rahmenbedingungen aufgrund von Nutzungsbeschränkungen und beziehungsweise oder Modifikationsbeschränkungen durch Lizenzeinschränkungen

c) Verlässlichkeit eines dekompilierten Toolkit

Liegt das Toolkit als Quellcode vor und nicht als voll funktional kompiliertes Produkt, gelten zusätzlich die folgenden Bedingungen:

Quellcode wurde in der Programmiersprache C# entwickelt Die zu einsetzende Entwicklungsplattform Xamarin basiert auf .NET und der Programmiersprache C#. [6, 56] Um eine grundsätzliche Kompilierbarkeit durch Xamarin zu gewährleisten, muss der Quellcode in der Programmiersprache C# vorliegen.

Quellcode enthält unterstützte .NET Befehlsmenge Neben dem Vorliegen des Toolkit-Quellcodes in der Programmiersprache C#, darf der Quellcode außerdem ausschließlich nur die .NET-Befehlsmenge enthalten, die sowohl von allen zu unterstützenden Zielplattformen für TheClient als auch von Xamarin selbst kompiliert werden können. Der Umgang mit .NET-Befehlsmengen wird in Kapitel 4.2 erläutert.

Quellcode ist frei von plattformspezifischen Befehlen Der Quellcode soll durch Xamarin nach Möglichkeit vollständig unabhängig von spezifischen Plattformfunktionalitäten auf den jeweiligen Plattformen verteilt werden. Die Verteilung des gleichen Quellcodes auf die unterschiedlichen Plattformen bedeutet gleichzeitig die zwingende Vermeidung plattformspezifischer Befehle in von der Plattform unabhängigem Quellcode. Konkret sollte der Toolkit Quellcode somit keine für eine bestimmte Plattform spezifischen Befehle enthalten.

Während die oben genannten Anforderungen an ein Toolkit zwingende Anforderungen sind, um die Lauffähigkeit innerhalb der Entwicklungsplattform Xamarin zu ermöglichen, wird nur eine als nachfolgende Anforderung als zusätzliche aber nicht unbedingt zwingende Anforderung identifiziert:

Toolkit muss dokumentiert sein Ein nicht oder unvollständig dokumentiertes Toolkit erfüllt den Sinn eines Toolkits nicht vollständig. Wie in den vergangenen Absätzen dieser Arbeit erwähnt, dient ein Toolkit neben der verkürzten Entwicklungszeit einer entsprechenden UA Anwendung auch zur Minimierung der Einarbeitungszeit. Ist ein Toolkit unvollständig oder nicht dokumentiert so muss die gewonnene Einarbeitungszeit in die mühsame Dokumentation investiert werden.

Es existiert zum aktuellen Zeitpunkt kein .NET Toolkit, welches als Quellcodevariante verfügbar ist. Abbildung 4.5 zeigt deshalb alle zum Zeitpunkt dieser Arbeit verfügbaren Toolkits als auch alle Stacks die auf .NET basieren. Toolkits und Stacks, die den aufgestellten Anforderungen entsprechen, sind in der Abbildung grün markiert, nicht erfüllte sind rot markiert und irrelevante Anforderungen von Toolkits beziehungsweise Stacks der Binärvariante sind grau

Toolkit / Stack	Hersteller	Modifizierbarer Quellcode	Ausreichend dokumentiert	Unterstützt notwendige .NET Befehlsmenge
.NET Based OPC UA Client SDK	Unified Automation	Binärcode		
OPC Foundation Unified Architecture .NET Reference Implementations	OPC Foundation			
OPC UA .NET Server und Client Development Toolkits	Softing Industrial Automation	Binärcode		
Client Development SDK for OPC UA	Technosoftware	Binärcode		
OPC-UA Framework Advanced Client and Server SDK for.NET	Traeger	Binärcode		
OPC UA Connector / UA Client SDK	Rothenbacher	Binärcode		
OPC UA .NET SDK	Prosys PMS	Binärcode		

Abbildung 4.5.: Auf dem Markt verfügbare .NET Open Platform Communications Unified Architecture Toolkits und Stacks

markiert. Toolkits und Stacks die als Binärversion vorliegen sind entsprechend der erwähnten Aspekte nicht ohne Modifikationsaufwand unter Xamarin lauffähig. Modifikationen durch Dekompilierung sind wiederum aufgrund von Nutzungsbeschränkungen und rechtlichen Beschränkungen nicht möglich.

Für die Implementierung von TheClient bleibt lediglich die Auswahl des Stacks der Referenzimplementierung der OPC Foundation, da dieser als einziger Stack in der Quellcodevariante vorliegt.

4.3.2. Fehlercodeanalyse des UA Stacks

Die aufgestellten Minimalanforderungen an ein Toolkit in Abschnitt 4.3.1 führten zur Erkenntnis, das kein Toolkit existiert, welches die Minimalanforderungen erfüllt. Mit dem UA Stack der OPC Foundation steht zumindest einen für TheClient als Grundlage zu verwendender Stack bereit, der einzelne Minimalanforderungen erfüllt. Durch den Umstand, dass selbst die Teilmenge eines Toolkits, mit dem Stack nicht alle Minimalanforderungen erfüllt, macht es notwendig zu Prüfen inwieweit der UA Stack modifiziert werden muss, um eine Lauffähigkeit zu erlangen. Hierzu wurde auf Basis der im Unterkapitel 4.2 gewählten Quellcodeverteilungsstrategie ein gemeinsames Projekt in Xamarin angelegt. Anschließend wurde das UA Stack in das gemeinsame Projekt inkludiert. Hierzu wurden nur die Dateien in die Kompilierumgebung inkludiert, welche auch in der *UA Core Library Solution* des UA Stacks inkludiert sind. Die *Build Actions* wurden nicht aus der Original Stack Lösung in der integrierten Entwicklungsumgebung *Visual Studio* übernommen, da einzelne *Build Actions* in Xamarin nicht verfügbar sind. Unter Umständen liegt die Fehlerzahl so etwas höher. Anschließend wurde der Quellcode für alle von Xamarin unterstützten Plattformen kompiliert. Die Abbildung 4.6 zeigt dabei die Fehler verursachenden Quellcodestellen für die jeweiligen Plattformen. Erst eine Behebung aller Fehler verursachenden Quellcodestellen für die zu unterstützenden Plattformen gewährleistet die Lauffähigkeit des Stacks und ermöglicht somit die Entwicklung der SDK Abstraktionsschicht und anschließend die eigentliche Entwicklung von TheClient.

Die Abbildung zeigt, dass eine Anpassung des UA Stack für iOS mit 8443 fehlerhaften Quellcodestellen und für Android mit 8441 fehlerhaften Quellcodestellen am meisten Arbeit verursacht, wenn hierbei die Fehlerquote als Kennzahl dient. Ebenfalls verursacht Windows Phone 8.1 mit 2398 Fehler verursachenden Quellcodestellen einen erheblichen Modifikationsaufwand. Mit 716 Fehler verursachenden Quellcodestellen bei UWP und 811 Fehler verursachenden Quellcodestellen bei Windows 8.1 ergeben sich für diese beiden Plattformen den geringsten Änderungsaufwand. Unabhängig von der Plattform betrachtet enthält das gemeinsame Projekt 9178 Fehler verursachende

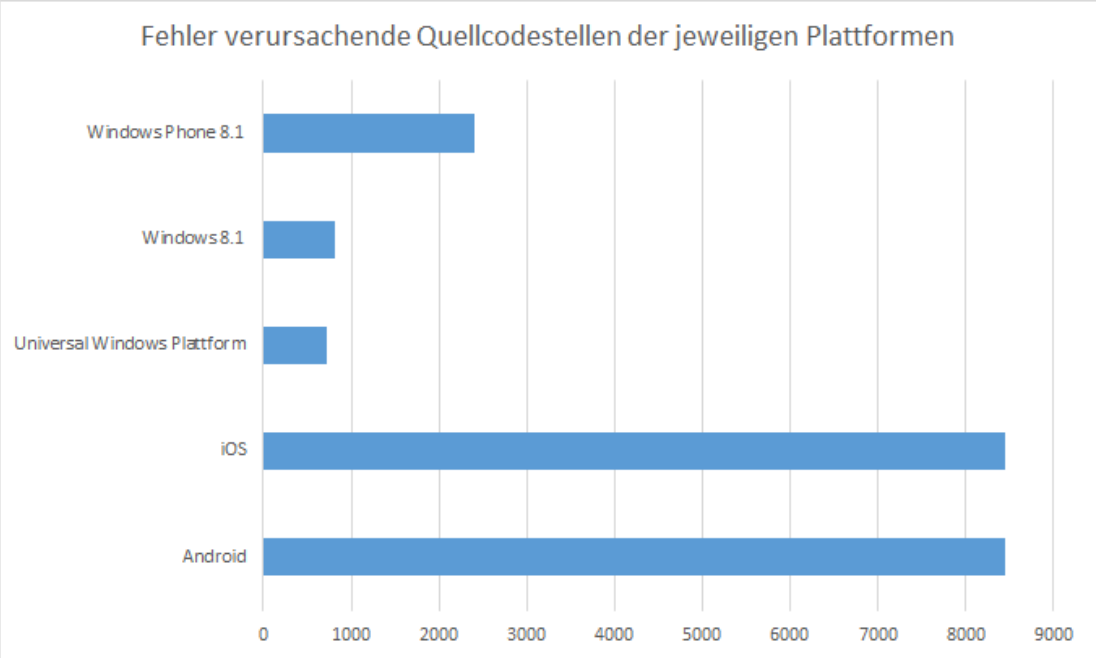

Abbildung 4.6.: Fehler verursachende Quellcodestellen der jeweiligen Plattformen

Quellcodestellen. 8664 der 9178 Fehler sind unmittelbar, auf in der jeweiligen Plattform nicht unterstützte und damit fehlende .NET-Typen zurückzuführen. Abbildung 4.7 zeigt, in welchen Dateien des UA Stacks sich die meisten Fehler verursachenden Quellcodestellen befinden. Insgesamt sind die fehlerhaften Quellcodestellen in 138 Dateien verteilt, von denen in der Abbildung nur diese ersichtlich sind mit einer Fehlerquote größer oder gleich hundert.

Mit 5648 ist die Datei `Opc.Ua.DataTypes` die größte Fehler verursachende Problemstelle. In dieser Datei sind alle für die Programmlogik nutzbaren Datentypen für das UA Informationsmodell beschrieben. Die hohe Relevanz der Datei kann aus diesem Grund nicht beschritten werden. 906 Fehler verursachenden Quellcodestellen enthält die Datei `Opc.Ua.Interfaces`, welche die Interfaces für die verschiedensten Methoden deklariert und als Schnittstelle zwischen Instanzen vermittelt. Die Datei `ApplicationConfiguration` enthält 632 Fehler verursachende Quellcodestellen. Die Datei ist für das Laden verschiedener für den Betrieb des UA Stacks notwendigen Konfigurationseinstellungen aus der Persistenzebene zuständig. Ebenfalls mit einer noch hohen Anzahl Fehler

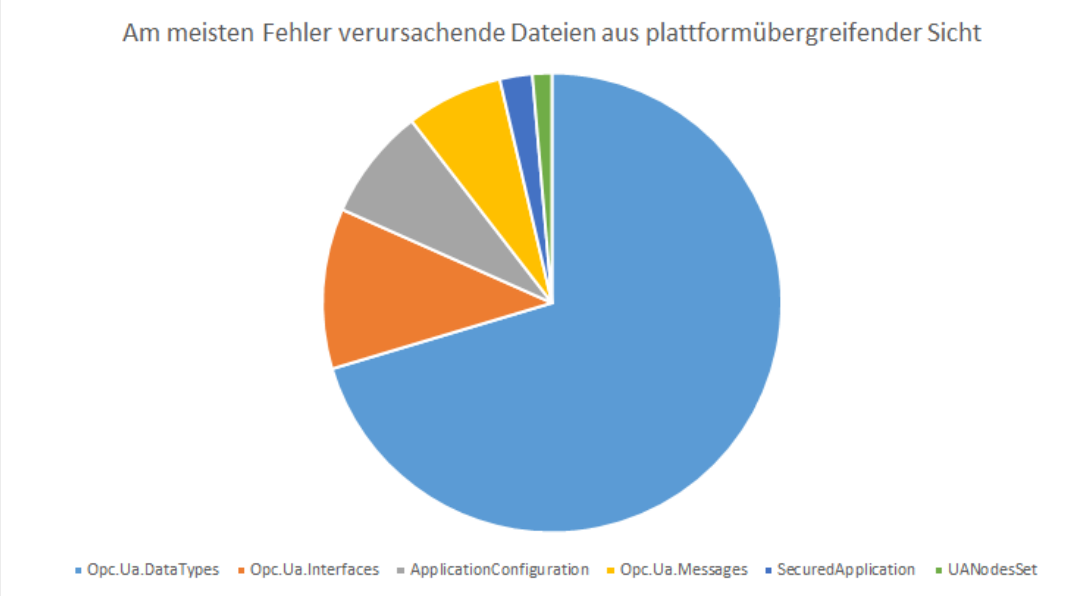

Abbildung 4.7.: Am meisten Fehler verursachende Dateien aus plattformübergreifender Sicht

verursachender Quellcodestellen ist die Datei `Opc.Ua.Messages` involviert. Die Datei ist für die Nachrichten zuständig, die durch die serviceorientierte Architektur zwischen UA Client und UA Server gesendet werden. 184 Fehler verursachende Quellcodestellen enthält die Datei `SecuredApplication` und noch 110 die Datei `UANodeSet`.

In Anhang C.1 findet sich die detaillierte Fehlercodeanalyse.

Während der prototypischen Implementierung veröffentlichte die OPC Foundation einen Stack, der speziell für UWP optimiert wurde. Die Analyse dieses Stacks ergab, dass dieser aufgrund der niedrigeren Fehler verursachenden Stellen geeigneter als Stack Grundlage für TheClient ist. Kapitel 5.3 beschreibt die Portierung dieses Stacks in ein Xamarin Projekt zur Entwicklung von TheClient. In Anhang C.2 findet sich die Fehlercodeanalyse des speziell für UWP optimierten Stacks.

4.3.3. Integrationsstrategie für den UA Stack

Ausgehend von der Analyse des UA Stacks ist zu erwarten, dass eine Modifizierung des Stacks erforderlich ist, um eine eigenständige Lauffähigkeit dieses Stacks zu gewährleisten, um im Folgenden die eigentliche Anwendungslogik zu implementieren. Vor der Modifizierung steht allerdings die Einbindung des Quellcodes des UA Stacks der OPC Foundation in das mit Xamarin nach Auswahl der Quellcodeverteilungsstrategie in Kapitel 4.2 angelegte gemeinsame Projekt. In Anbetracht der durchzuführenden Modifikationen von integriertem Quellcode des UA Stacks ergeben sich die folgenden beiden Strategien zur Handhabung solchen Quellcodes.

Datei für Datei Transfer Bei dieser Variante würde Datei für Datei in das Projekt implementiert werden. Bei jedem Dateizuwachs würde die Datei auf Abhängigkeit zu anderen C#-Typen oder Referenzen anderer Dateiarten des UA Stacks geprüft werden. Besteht eine solche Abhängigkeit zwischen der zu implementierenden und der bei Betrachtung in Beziehung stehenden Datei, würde diese ebenfalls in das Projekt implementiert werden. Dateien die Referenzen zur .NET-Bibliothek pflegen, die aufgrund der für Xamarin oder einer speziellen im TheClient Projekt unterstützten Plattform die entsprechende .NET-Teilmenge gemäß Kapitel 4.2 nicht unterstützen, werden somit direkt identifiziert und können durch entsprechende Modifikationen zur Kompilierbarkeit und im späteren, nach der Implementierung aller nötigen Dateien, zur Lauffähigkeit gebracht werden. Die Variante ermöglicht so eine schrittweise Portierung des Quellcodes in das TheClient Projekt unter direkter Erkenntnis, welche Dateien und insbesondere welche Stellen innerhalb der Dateien eine Unabhängigkeit von der Plattform verhindern. Vorteile, die für diese Variante sprechen ist die intensive Befassung mit dem UA Quellcode, welche die fehlende Dokumentation des UA Stack Quellcodes ersetzen kann. Weiterhin kann durch dieses Vorgehen ein Fehler nach dem anderen behoben werden und eine grundsätzliche Kompilierbarkeit des Quellcodes ist dadurch schneller zu erreichen. Die Nachteile dieser Variante sind die sehr zeitintensive Portierung des Quellcodes und die schnellere Implementierung von Fehlern während der

Modifikation von für die Plattformunabhängigkeit gefährdende Quellcodestellen.

Vollständiger Transfer Bei der Variante des vollständigen Transfers würde im Gegensatz zum Datei für Datei Transfer, die vollständigen Dateien des UA Stacks in das gemeinsame Projekt inkludiert werden. Damit sind bereits alle Referenzen aufgelöst und es müssen sich nur noch die Probleme für die Plattformunabhängigkeit verursachenden Quellcodestellen angesehen werden. Durch den Wegfall der Auflösung von Referenzen kann der UA Stack wesentlich schneller in das Projekt inkludiert werden. Dies bedeutet bei ersterer Betrachtung einen zeitlichen Vorteil gegenüber der Variante bei der Datei für Datei transferiert und jeweils die Referenzen aufgelöst werden müssen. Gleichzeitig erfährt der Entwickler beim vollständigen Transfer anders als bei der ersten Variante kein Wissenszuwachs. Dies bedeutet für die Behebung einer für die Plattformunabhängigkeit verursachende Quellcodeproblemstelle eine längere Behebungszeit durch fehlendes Kontextwissen des Entwicklers. Ein wesentlicher Vorteil dieser Variante gegenüber des Transfers Datei für Datei ist allerdings die Vermeidung von Fehlern bei der Auflösung von Referenzen.

Es ist festzustellen, dass bei einem intensiv dokumentieren Quellcode die Variante des vollständigen Transfers der Variante bei der Datei für Datei transferiert wird, immer vorzuziehen ist. Denn der Kontext, der zur Lösung Problem verursachender Quellcodestellen von Relevanz ist, kann in der Dokumentation nachgelesen werden. Ist eine solche Dokumentation, wie das beim gewählten UA Stack der Fall ist, nicht oder nur in rudimentärer Ausführung vorhanden, ist die Wahl der Integrationsstrategie abhängig der Problem verursachenden Quellcodestellen. Enthält der UA Stack nur eine geringe Anzahl an Problem verursachenden Quellcodestellen, kann die Variante des vollständigen Transfers der Variante bei der Datei für Datei transferiert wird vorgezogen werden, da der zeitliche Vorteil bei Einsparung von nicht aufzulösenden Referenzen, der Wissensaufnahme zum Verständnis der Quellcodestelle nicht überwiegt. Ist die Anzahl allerdings hoch, kann mit der Variante des Datei für Datei Transfers bereits bei der Auflösung der Referenzen die Wissensaufnahme angeeignet werden, die für die Behebung der Problem verursachenden Quellcodestelle

benötigt wird. Zur Entscheidungsfindung, der für TheClient möglichst zeitsparender und gleichzeitig möglich Fehler Isolierenter Variante wurde die möglichen Fehler verursachenden Quellcodestellen identifiziert. Aufgrund der hohen Anzahl von in Anhang C identifizierten Fehler verursachenden Quellcodestellen wird die Variante des Datei für Datei Transfers für die Implementierung von Client gewählt.

4.3.4. Modifizierungsstrategie für den UA Stack

Nachdem der Quellcode des UA Stacks durch die in Kapitel 4.3.3 gewählte Integrationsstrategie in die Anwendung TheClient integriert wurde, stellt sich die Frage wie der betroffene Quellcode, der einer Änderung bedarf modifiziert werden muss zur Beibehaltung der Architektur- und Methodenstruktur des Stacks. Dies ist deshalb von Relevanz, da eine Änderung der Grundstruktur des Stacks zur Folge hat, dass Updates des Open Source Stacks durch die Community nicht mehr einfach im modifizierten Stack in TheClient durchzuführen sind. Die Signifikanz einer Modifizierungsstrategie steigt bei Betrachtung der zum Zeitpunkt der Anfertigung dieser Arbeit des schnell sich verändernden Quellcodes durch die Open-Source-Gemeinschaft. Diese Änderungen beheben derzeit akute Fehler im Stack, von denen TheClient auch als prototypische Anwendung profitieren soll.

Es ergeben sich hierdurch die folgenden Regeln bei Modifizierung des originalen portierten Quellcodes:

1. Lagere immer die gesamte Methode, in der ein Fehler verursacht wird, in das spezifische Projekt aus
2. Teile Methoden nicht auf
3. Ändere den Methodenkopf nicht

Die aufgestellten Regeln zur Modifizierung sollten als Richtlinie, aber keinesfalls als zwingend erachtet werden. Damit kann in Ausnahmefällen von der Richtlinie wenn nötig abgewichen werden. Exemplarisch sei hier der Fall zu erwähnen, bei dem die Fehleranzahl in einer Methode überschaubar ist und

ein Umschichten der Methode in ein spezifisches Plattformprojekt zusätzlich ein größeres Umschichten von referenzierten Methoden und Klassenattribute zur Folge hätte. In solchen Fällen kann in TheClient ausschließlich die Fehler verursachende Quellcodestelle umgeschichtet werden.

Die Modifizierung nach den aufgestellten Regeln wird dabei immer nach demselben Schema praktiziert:

1. Der Methodenkopf der Methode, in welcher der Fehler verursacht wird, wird in einem entsprechenden gemeinsamen Interface hinterlegt. Falls nur einzelne Codestellen transferiert werden, wird ein entsprechender Methodenkopf im Interface erzeugt.

2. Jedes plattformspezifische Projekt implementiert das entsprechende Interface

3. Der originale Quellcode geht von der Methode im Shared Projekt in die Methode, die durch das Interface im plattformspezifischen Projekt angelegt wurde

4. Der originale Quellcode in der Methode der spezifischen Plattform wird so modifiziert, dass er für die Plattform funktionsfähig ist

5. Der nun leere Methodenrumpf der Methode beziehungsweise die leere Codestelle im Shared Projekt wird gefüllt durch Aufruf der Methode über das gemeinsame Interface

4.4. Datenvisualisierung

Mit einer erfolgreichen Implementierung des UA Stacks und des SDKs kann der Datenabruf und die Datenmodifizierung gemäß der Forschungsfrage nachgewiesen werden. Zur Nachweiserbringung fehlt allerdings noch die in der Forschungsfrage definierte Datenvisualisierung. Die nachfolgenden Abschnitte beschreiben die Konzeption einer exemplarische Abbildung von Daten aus dem UA Informationsmodell in eine grafische Benutzeroberfläche.

Der folgende Abschnitt zeigt den Kontext der Anwendung TheClient auf, in welcher dieser im Produktiveinsatz zum tragen käme. Die Relevanz der im folgenden Abschnitt gewonnenen Erkenntnisse und Informationen zeigt sich bei der Entscheidungsfindung für die in der Forschungsfrage definierte Visualisierung. Zur Beschreibung des Kontexts dienen die charakteristischen Eigenschaften sowie Bedürfnisse der zukünftigen Anwender der Anwendung. [62] Die Bedürfnisse der Anwender lassen sich aus den konkreten Anwendungsfällen aus Unterkapitel 3.3 ableiten. Informationen zu charakteristischen Eigenschaften zukünftiger Anwender lassen sich hingegen durch Studieren öffentlicher Informationsquellen bewerkstelligen.

Bei der industriellen Branchenzielgruppe handelt es sich um ein Männer dominiertes Arbeitsgebiet. [63, 64] Nur ein Anteil von 16,6 Prozent aller weiblichen Berufstätigen ist im produzierenden Gewerbe tätig. [64] Das Durchschnittsalter zukünftiger Anwender beider Geschlechter liegt bei 42 Jahren. Die Wahrscheinlichkeit, dass eine Frau unter 25 Jahre alt ist oder über 50 Jahre liegt bei der Gruppe aller erwerbstätigen Frauen bei 48 Prozent. Die Gruppe von Personen, die kurz vor dem Renteneintrittsalter oder das Renteneintrittsalter bereits überschritten haben ist überproportional von Männern dominiert. Europäisch betrachtet zählt die Zielgruppe der erwerbstätigen deutschen Männer zu den höchsten. [63]

Ein wichtiger Aspekt nach der Eingrenzung der Gestaltungszielgruppe ist die Definition der Bestandteile einer Benutzeroberfläche. Die grafische Benutzeroberfläche dient als Wirt für die eigentliche Visualisierung der Daten und kann durch die von Brad Frost [65] entwickelte Methodik *Atomic Design* beschrieben werden.

Die Methodik des Atomic Design zur Beschreibung eines Designs identifiziert fünf Bestandteile einer grafischen Benutzeroberfläche, die Begriffe aus der chemischen Theorie verwenden: Atome, Moleküle, Organismen, Vorlagen und Seiten. Ein Atom beschreibt die kleinste Einheit eines Designs. Es darf nicht weiter zerteilt werden, um dessen Funktion nicht zu verlieren. Hierbei enthält ein Atom eindeutige Eigenschaften und kann abstrakte Elemente wie beispielsweise Animationen oder eine Farbpalette beinhalten. Beispiele für Atome

im Design können Schaltflächen, Eingabefelder oder Beschriftungen mit einer festen Größe als Eigenschaft sein. Die nächste größere Einheit im Design ist nach dem Atom das Molekül. Ein Molekül ist eine Menge von Atomen, die gemeinsam erst einen Zweck erschaffen. Beispielsweise kann die Beschriftung mit dem Eingabefeld kombiniert werden. So ergibt sich aus zwei Atomen ein Molekül, das für das Eingabefeld den Zweck erzeugt, in dem der Anwender durch die Beschriftung nun weiß, für welche Informationsaufnahme das entsprechende Eingabefeld ist. Die Moleküle wiederum können in einem größeren Kontext, den Organismen, abgebildet werden. Ein Exemplar für einen Organismus im Design wäre der Kopfbereich einer Anwendung. Mit den Vorlagen werden die einzelnen Komponenten platziert und die eigentliche Inhaltsstruktur gegliedert. Seiten hingegen sind spezifische Instanzen der Vorlagen. [65]

4.4.1. Informationsarchitektur und Seitenstruktur

Die Informationsarchitektur dient als Basis für den Visualisierungsaspekt der Forschungsfrage. Eine Informationsarchitektur beschreibt hierbei, wie Informationen geordnet, strukturiert und bezeichnet werden müssen um die Ziele des Anwenders zum Abruf und zur Modifizierung von Gerätedaten optimal zu unterstützen und ein angenehmes Nutzungserlebnis zu ermöglichen. [66, 67] Die Relevanz zur Strukturierung von Daten im Rahmen des Visualisierungsaspekts mit der Informationsarchitektur steigt dabei zusätzlich bei Betrachtung der theoretisch unendlich abzurufenden Daten aus dem UA Informationsmodell. [67] Ziel dieses Kapitels ist die Identifizierung und Strukturierung von den aus dem UA Informationsmodell abzurufenden Informationen zur Planung der Seitenstruktur, um mit dieser Struktur für die Anwendung die Erreichung der möglichst einfachen Auffindung vom Anwender gesuchten Daten aus dem UA Informationsmodell zu erreichen. [66, 67] Der Autor Moser [67] empfiehlt als ersten Schritt zur Entwicklung der Informationsarchitektur eine Bestandsaufnahme der zu, visualisierenden Informationen durchzuführen. Dies bedeutet nicht, dass jede einzelne Information die für die Anwendungsvisualisierung relevant sein könnte, zu identifizieren ist. Vielmehr bedeutet es die Identifikation

der verschiedenen Arten von Informationen, die durch die für TheClient festgelegten Anforderungen extrahiert werden können. Nur die möglichst vollständige Identifikation in TheClient zu visualisierenden Informationen macht die Planung der grafischen Benutzeroberfläche möglich. Informationsarten können auf zwei Wegen aus den Anforderungen in Kapitel 3.3 extrahiert werden. Entweder ist eine Quelle zu identifizieren aus denen die Informationsarten extrahiert werden können oder aber, wenn keine Quelle zu identifizieren ist, müssen die Informationsarten durch explizites Wissen durch Betrachtung des realen Abbilds der Welt erkannt werden. Exemplarisch könnte so für die Anforderung aus Kapitel 3.3.3 zum Lesen eines Datenknotens der reale Weltgegenstand *Buch* (extrahiert aus der Abbildung Datenknoten => Datenobjekt => zum Beispiel Buch) betrachtet werden. Mit dem expliziten Wissen, das ein Mensch über das Lesen eins Buches hat, kann der Gedankengang angestrengt werden, was benötigt wird, um ein Buch zu lesen. Die Informationen, die sich aus dem expliziten Wissen herausfiltern lassen können auf die Anforderung zum Lesen eines Datenknotens übertragen werden. Beispielsweise kann man mit dem expliziten Wissen identifizieren, dass es notwendig ist, um ein Buch zu lesen, den Namen des Buches, das man Lesen möchte zu kennen. Überträgt man dieses Wissen auf die Anforderung, so erhält man für die Bestandsaufnahme, die Information, dass die Identifikation des Datenknotens bekannt sein muss, um ihn zu lesen. Das Beispiel zeigt, dass sich Informationen für die Bestandsaufnahme aus den Anforderungen für welche keine Quelle identifiziert werden kann, aus dem expliziten Wissen eines Menschen extrahieren lassen.

Für die Entwicklung von TheClient lassen sich alle Informationen der in Kapitel 3.3 definierten Anforderungen für TheClient unmittelbar aus Quellen beziehen. Denn die definierten Anforderungen sind immediat auf die Anwendungsfälle der UA Technologie zurückzuführen und diese wiederum wurden ursprünglich, wie Abbildung 3.1 aus Kapitel 3.1.1 zeigt, aus den Diensten der UA Technologie gewonnen. Somit können die Informationen direkt aus den im UA Standard detailliert spezifizierten Diensten extrahiert werden. Ein Dienst gemäß einer serviceorientierten Architektur wird durch Informationen (Auftragsdaten) initiiert und produziert als Resultat wiederum Informationen (Antwortdaten).

Anfrage- und Antwortdaten sind somit grundsätzlich für TheClient von Relevanz. Anfragedaten sind sogar zwingend notwendig zum Aufruf des Dienstes und damit zur Erfüllung der Anforderung. Um Informationen für die Erstellung der Seitenstruktur möglichst transparent zu erfassen, werden diese soweit wie möglich unabhängig von anderen Daten notiert, ohne den Kontext der Information zu verlieren. Zusammenhängende Informationen, welche ohne Verknüpfung zueinander unvollständig oder fehlerhaft sind - oder fataler: den Kontext verändern -, werden also nicht voneinander getrennt.

Nachfolgend findet sich eine Auflistung aller für die Erfüllung der jeweiligen Anforderungen beitragenden aber nicht notwendigerweise zu visualisierenden Informationen, die aber zu mindestens eine Sinnhaftigkeit bei der Anzeige in der grafischen Benutzeroberfläche hätten. Welche Informationen dann von der Bestandsaufnahme der Informationen tatsächlich in der grafischen Benutzeroberfläche visualisiert werden, wird bei der Erfassung der Seitenstruktur definiert. Informationen, die nicht zur Kernerfüllung der Anforderung beitragen oder auf direkt ersichtlicher Irrelevanz basieren werden in der Bestandsaufnahme nicht erfasst. Informationen, die durch API Aufrufe im UA Stack abstrahiert sind, gehen ebenfalls nicht in die Bestandsaufnahme ein. Die Bestandsaufnahme einer Information führt nicht zwingend zur Einbindung dieser Information in der Informationsarchitektur.

Die in Kapitel 3.3.1 definierten Anforderungen um UA Server in der administrativen Domäne zu finden und sich mit diesen zu verbinden verwenden von UA definierte Anforderungen, die an folgenden UA Dienstgruppierungen gemäß Tabelle 3.1 gebunden sind: Discovery Service Set [36], Secure Channel Service Set [36] und Session Service Set [36]. Daraus ergibt sich nach Bestandsaufnahme die folgende vorgelegte Sammlung an Informationen:

- Discovery URL
- Discovery Server URL
- Globaler Identifikator einer UA Anwendung
- Anzeigbarer Anwendungsname

- Anwendungstyp der UA Anwendung (Server, Client, Server & Client, Discovery Server)
- Sitzungsendpunkte einer UA Anwendung
- Nachrichtenübertragung Sicherheitsmodus (Ohne, Signiert, Signiert & Verschlüsselt)
- Benutzeridentifikationsarten
- Lebensdauer des sicheren virtuellen Kanals
- Identifikator des sicheren virtuellen Kanals
- Sitzungsname
- Beschreibung der UA Client Anwendung
- Sitzungszeitüberschreitung
- Identifikator der Sitzung
- Löschung von an Sitzung gebundene Abonnements
- Anzahl abgebrochener Dienstanfragen bei Beendigung einer Sitzung

In Kapitel 3.3.2 werden die Anforderungen spezifiziert, welche es erlauben Datenknoten in einem Adressraum zu durchsuchen. Durch die Bestandsaufnahme ergeben sich für diese Anwendungsfälle die folgenden relevanten Informationen (View Service Set, Query Service Set):

- Anzahl anzuzeigender Datenknoten
- NodeClass
- Identifikator des Datenknoten
- Datenknoten Arbeitstitel
- Anzeigbarer sprachabhängiger Name des Datenknoten

Für die definierten Anforderungen in Kapitel 3.3.3 gelten die folgenden relevanten Informationen (Read Service):

- Cached Werte erlauben
- Identifikator des Datenknoten
- Identifikator des Datenknoten Attributs
- Wert
- Status Code
- Zeitstempel der letzte Parameteränderung

Für die definierten Anforderungen in Kapitel 3.3.4 gelten die folgenden relevanten Informationen (Write Service):

- Identifikator des Datenknoten
- Identifikator des Datenknoten Attributs
- Wert
- Status Code
- Zeitstempel der letzten Parameteränderung

Für die definierten Anforderungen in Kapitel 3.3.5 gelten die folgenden relevanten Informationen (Subscription Service Set, Monitored Item Service Set):

- Publishing Intervall für das Abonnement
- Publishing aktiv/inaktiv
- Priorität in relativer Abhängigkeit zu anderem vom Client erzeugten Abonnements
- Identifikator des Abonnements
- Identifikator des Datenknoten
- Identifikator des Attributs
- Modus des überwachenden Element (Deaktiviert, Sampling, Reporting)
- Identifikator des zu überwachenden Element

- Millisekunden in denen nach Änderungen geprüft wird
- Maximale Anzahl an Elementen in der Publishing Warteschlange
- Typ des Nachrichtentriggers für Element (Status des Wertes, Wert oder Status des Wertes (Standard), Quellzeitstempel oder Wert oder Status des Wertes)
- Deadband aktiv/inaktiv
- Deadband (Absolut, Prozentual)

Für die definierten Anforderungen in Kapitel 3.3.6 gelten die folgenden relevanten Informationen:

- Anzeigbarer sprachabhängiger Name der Topologie-Datenknoten

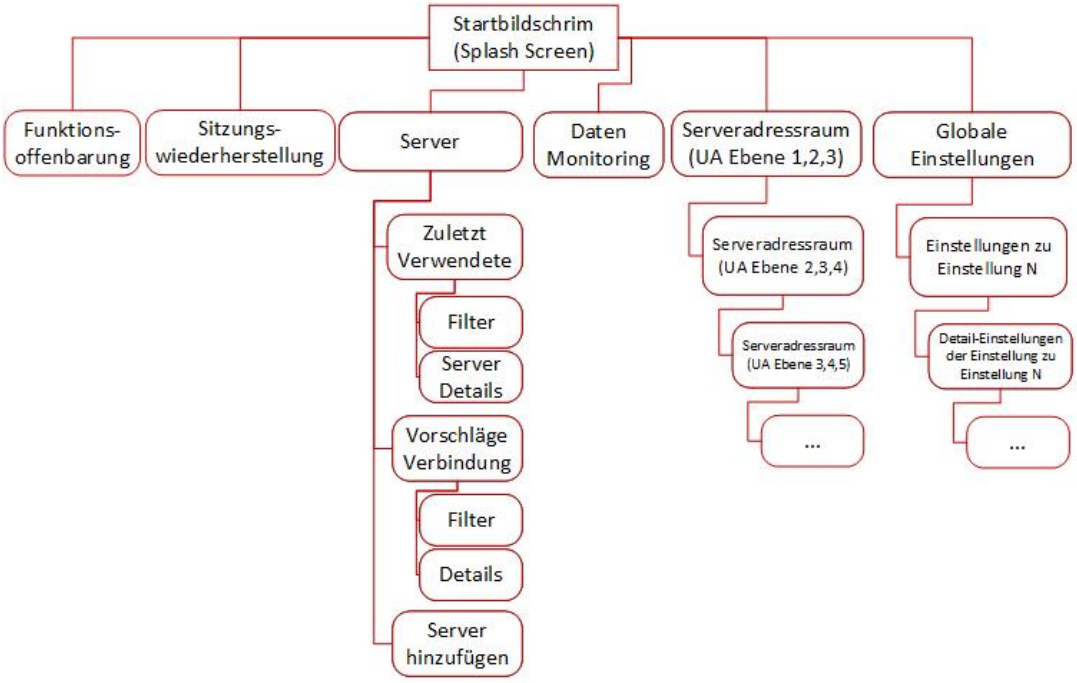

Abbildung 4.8.: Informationsarchitektur für TheClient

Die ermittelten Informationen lassen sich zu der in Abbildung 4.8 illustrierten Informationsarchitektur strukturieren. Die Informationsarchitektur für TheClient

orientiert sich stark an den in Kapitel 3.3 identifizierten Anforderungen. Die Informationen aus der Bestandsaufnahme können so direkt den einzelnen Knoten der Informationsarchitektur zugeordnet werden.

Die Informationsarchitektur muss in der Praxis nicht mit der Seitenstruktur der grafischen Benutzeroberfläche übereinstimmen. [67] In diesem Fall ist die Informationsarchitektur für die Realisierung von TheClient passend und wird gleichzeitig als Seitenstruktur verwendet. Die Anwendung basiert somit auf drei Hauptseiten: Server, Daten Monitoring und Serveradressraum. Direkt nach Start der Anwendung ist die Seite des Daten Monitoring sichtbar. Von dieser Seite kann direkt zum Server oder zum Serveradressraum navigiert werden. Die globalen Einstellungen sind von jeder Seite zu erreichen. Die Sitzungswiederherstellung nur beim Start der Anwendung, wenn eine Sitzung zu wiederherstellen ist. Die Funktionsoffenbarung ist bei der aller ersten Verwendung der Anwendung sichtbar.

Die meisten Inhalte lassen sich in der grafischen Benutzeroberfläche von TheClient statisch repräsentieren. Für die Informationen, die aus dem UA Informationsmodell entnommen werden, ist dies nicht möglich. Wie in den vorherigen Kapiteln erläutert, wurde UA mit dem Ziel der Unabhängigkeit konzipiert. Durch dieses Ziel können im Informationsmodell, Daten beliebigen Typs eingebunden werden und durch den UA Client gelesen oder aktualisiert werden. Für die grafische Benutzeroberfläche ergibt sich somit eine ebenfalls theoretisch unendliche Menge an Möglichkeiten zur Darstellung der verschiedenen Informationstypen. So kann exemplarisch eine Zahl, die im Informationsmodell nur mit Leseberechtigung charakterisiert wurde anders in der Benutzeroberfläche dargestellt werden, als eine Menge begrenzter Modifizierungsmöglichkeiten für einen Datenknoten, für den genau eine Modifizierungsmöglichkeit vom Anwender gewählt werden muss. Die Basis die eine Visualisierung gemäß der Forschungsfrage ermöglicht, ist somit die dynamische Abbildung der nicht-statischen Informationen - konkret: Informationen aus dem UA Informationsmodell. Die besondere Signifikanz der dynamischen Informationsvisualisierung wird nochmals deutlich mit dem Wissen, dass Standardisierungsorganisationen, eigene Informationsmodelle definieren können. Würde die Informationsvisualisierung also nicht dynamisch ermöglicht werden, wür-

de man für die häufigsten Informationstypen, statische Benutzeroberflächen-Komponenten definieren. Exemplarisch könnte man dann die Programmlogik so definieren, dass Knoten X durch die Benutzeroberflächen-Komponente Y dargestellt wird. Wie bereits erwähnt, könnten so zwar die wichtigsten Informationen in der Benutzeroberfläche abgebildet werden, würde allerdings das Informationsmodell von der Struktur verändert werden oder andere Informationsmodelle, wie das *OPC UA Informationsmodell für FDT*, würde mit einbezogen werden, müsste die Programmlogik angepasst werden. Das folgende Kapitel 4.4.2 definiert das theoretische Konzept der dynamischen Informationsvisualisierung.

4.4.2. Dynamische Informationsvisualisierung

Während Forschungsarbeiten wie die von Jeffrey Nichols [68] oder Krzysztof Z. Gajos [69] zeigen, wie es möglich ist, die vollständige Benutzeroberfläche dynamisch generieren zu lassen, ist dies für die dynamische Informationsvisualisierung nicht notwendig. Denn lediglich die dynamischen Inhalte, wie die aus dem UA Informationsmodell, sollen dynamisch visualisiert werden. Dynamische Inhalte aus dem UA Informationsmodell sind wie in Kapitel 4.4.1 erläutert, solche, von denen der Typ der Information der Programmlogik für den Aufbau der grafischen Benutzeroberfläche nicht bekannt ist. Zur Betrachtung, welche Teile der grafischen Benutzeroberfläche dabei dynamisch generiert werden müssen und welche statisch definiert werden können, dient das in Kapitel 4.4 definierte Modell des Atomic Design.

Vorlagen entsprechend dem Atomic Design können für die grafische Benutzeroberfläche von TheClient statisch entwickelt werden. Die Vorlagen geben den Rahmen, in welchen die Informationen, welche dynamisch sein können, visualisiert werden. Eine dynamische Generierung dieses Rahmens ist nicht notwendig, sondern kann durch die in Kapitel 4.4.4 definierten visuellen Aspekten umgesetzt werden. Jede Seite in der in Kapitel 4.4.1 definierten Seitenstruktur ist äquivalent zu genau einer Vorlage und damit in Referenz zu der Atomic Design Methodologie, äquivalent zu genau einer Instanz genau einer Vorlage.

Somit gibt es exemplarisch eine Vorlage, welche als Rahmen für die dynamischen Informationen des UA Adressraumes dient. Wichtig in Referenz zum genannten Beispiel ist hierbei, dass die Vorlage so konzipiert wurde, dass die hierarchische Informationsstruktur der UA Datenknotenabbildung, keine Beziehung zu einem bestimmten Datentyp visualisiert. Das würde ansonsten bedeuten, dass die Vorlage nicht vollständig unabhängig vom Informationstyp ist. Ein Hierarchieknoten innerhalb der grafischen Informationsstruktur wird hingegen dynamisch generiert werden müssen, muss aber für die Vorlage irrelevant sein.

Die Konzeptgegenstände Atome und Moleküle der Atomic Design Methodologie sind gegenüber Vorlagen wie bereits verdeutlicht zu Entwicklungszeit nicht per se bekannt oder müssen wie am Beispiel des UA Informationsmodell, aufgrund der theoretisch unendlichen Typen von Informationen, dynamisch abgebildet werden. Konkret bedeutet das für die Implementierung von TheClient, dass die Typen der Informationen aus dem UA Informationsmodell erst zur Laufzeit bekannt werden. Die Auswahl der passenden Benutzeroberflächen-Komponente, wie beispielsweise eine Schaltfläche oder ein Eingabefeld, ist dann abhängig des Informationstyps und der Information selbst. Die Schaltfläche würde exemplarisch in der grafischen Benutzeroberfläche von TheClient also dann angezeigt werden, wenn es sich beim Informationstyp des UA Datenknotens, um eine Methode handelt. Die Schaltfläche würde somit die Methodenausführung repräsentieren und diese bei Betätigung anstoßen. Die Bindung der einzelnen Atome zu den Molekülen, wie in der Atomic Design Methodologie erläutert, ergibt sich durch die Datenknoten des UA Informationsmodells. So stellt der Datenknoten den Kontext und damit das Molekül dar, während die einzelnen Informationen innerhalb des Datenknotens die Atome für den Kontext und damit das Molekül repräsentieren. So ergibt sich exemplarisch aus den beiden Atomen *Eingabefeld*, der den Wert des Datenknotens repräsentiert, und dem *Anzeigetext*, der den Anzeigenamen des Datenknotens repräsentiert, ein Molekül. Erst die Zusammensetzung der beiden exemplarischen Atome als Molekül ergibt für den Anwender einen Sinn, dadurch, dass dem Anwender sofort ersichtlich ist, welche Daten das Eingabefeld repräsentiert.

4.4.3. Entscheidungslogik für die dynamische Informationsvisualisierung

Die Entscheidungslogik führt durch vorher definierte Regeln zur Auswahl einer Benutzeroberflächen-Komponente. Hierzu muss für jede Benutzeroberflächen-Komponente, die zur Nutzung in der grafischen Benutzeroberfläche verwendet werden soll, vorher die Bedingungen definiert werden, welche zur Auswahl der entsprechenden Komponente führen. Nach der Auswahl einer Komponente wird diese, wie in Kapitel 4.4.2 erläutert, entsprechend mit der Informationsstruktur einer spezifischen Vorlage zusammengeführt.

Dabei soll das Konzept der Entscheidungslogik so dynamisch definiert werden, dass nicht zwingend die Erfüllung aller Regeln zur Auswahl einer Komponente führt. Vielmehr soll durch den Satz von Regeln, welche vollständig oder zum Teil erfüllt werden, die den definierten Regeln am ehesten entsprechende Komponente zur Anzeige ausgewählt werden.

Ein Beispielsatz an Bedingungen, der aufgrund der Informationen eines Datenknotens zur Auswahl eines An/Aus-Schalters führt, kann wie folgt lauten:

1. Der Wert ist eine Zahl.
2. Der Wert ist veränderbar.
3. Der Wert liegt im Wertebereich zwischen und einschließlich 0 und 1.

Durch Kombination des An/Aus-Schalters mit beispielsweise dem Anzeigenamen des Datenknotens ergibt sich ein Molekül, welches exemplarisch in einem vertikalen Layout, dem Organismus angezeigt werden kann. Dadurch ergibt sich die entsprechende Benutzeroberflächen-Komponente, welche für das Anzeigen dieses bestimmten Datenknotens aus dem UA-Informationsmodells innerhalb der Benutzeroberflächen-Vorlage verwendet werden kann.

Die Bedingungen zur Auswahl der Komponenten ergeben sich durch die Datenknoten und insbesondere durch die Metainformationen innerhalb der Datenknoten. Im Prototyp wir die dynamische Auswahl der Komponenten auf Grundlage des Wertes der Datenknoten beschränkt, mit der die Visualisierung

nachweisbar wird. Die Menge der Bedingungen findet sich als Implementierung durch Quellcode auf beiliegender Digital Versatile Disc (DVD).

4.4.4. Visuelle Gestaltung

Die visuelle Gestaltung befasst sich mit dem in der Forschungsfrage definierten Visualisierungsaspekt. Konkret mit der Gestaltung der Benutzerschnittstelle. Hierzu gehören die Wahl von Farben, Formen, Schriften, Symbolen und Bildern sowie das Definieren von Rastern, Abständen und Anordnungen. Dies verleiht der Anwendung eine Identität und Wiedererkennbarkeit. Farben, Formen und Bilder wecken Emotionen, welche die Stimmung und Erlebnis intensivieren. So kann bewusst die Aufmerksamkeit des Anwenders gesteuert werden. [70] Grundlage für die zu treffenden Designentscheidungen ist die in Kapitel 4.4 beschriebene Zielgruppe. Der grundlegende zu beachtende Aspekt bei der Auswahl eines für die Anwendung geeigneten Farbschemas und einer passenden Typografie war die Erkenntnis, dass der Anwender die Anwendung in der Regel mehrere Stunden ohne größere Unterbrechungen verwenden wird. Ausgehend von dieser These, ist die Wahl von kontrasthaltigen Farben und einer möglichst großen Schriftgröße jeweils zur Schonung der Augen als sinnvoll zu betrachten. Ein weiterer Aspekt ist die Vermeidung von Farben, die aufgrund von Farbfehlsichtigkeiten nicht unterschieden werden können. Als Beispiel sei hier die 'Rot-Grün-Sehschwäche' zu erwähnen. Als Grundlage für die Auswahl des Farbschemas diente die von der Firma Scott Design Inc durchgeführte Umfrage [71]. Die Umfrage ermittelt, welche Farbtöne den Menschen am meisten gefallen und welche am wenigsten gemocht werden. Dabei gruppiert die Umfrage die Teilnehmer nach Kultur, Alter und Geschlecht. Da die Umfrage nicht zeitlich begrenzt ist, werden fortlaufend seit November 2011 Ergebnisse gesammelt. Mit Befragung von über 3000 Personen (Stand Juni 2016) aus über 120 Nationen gilt die Erhebung als eine der größten. Wie Abbildung 4.9 zeigt, sind die Farben blau und grün die Lieblingsfarben der meisten Personen, während braun und gelb am wenigsten gerne gesehen werden. [72]

Ausgehend von dieser Datenerhebung werden die Farben Blau, Grün und Rot näher auf ihre psychologischen Eigenschaften und Wirkungen analysiert.

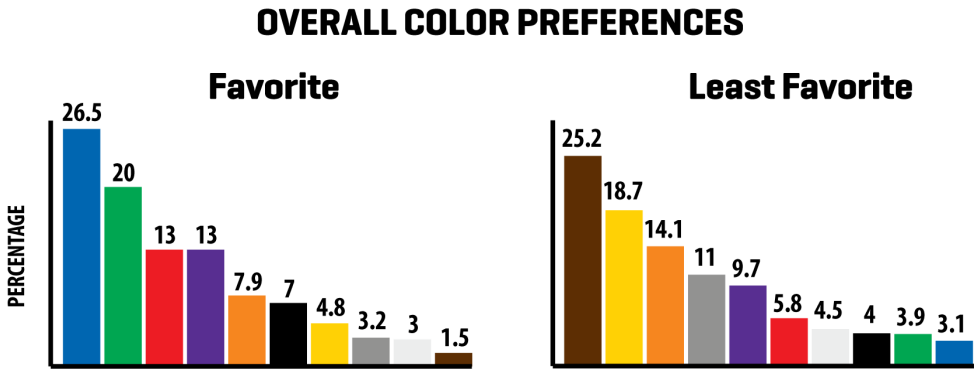

Abbildung 4.9.: Von Menschen bevorzugte und weniger bevorzugte Farben [72]

Nach Erkenntnissen der Farbpsychologie vermittelt die Farbe blau Intelligenz, Vertrauen, Sicherheit, Kreativität und Ruhe. [73, 74] Während helle Blautöne eher Gesundheit, Heilung, Ruhe, Verständnis und Sanftheit vermitteln, vermitteln dunkle Blautöne Wissen, Power, Integrität und Professionalität. [73] Blau vermittelt außerdem Männlichkeit und wird von Männern bevorzugt, die mit 57 Prozent die Lieblingsfarbe von Männern ist. Bei Frauen liegt der prozentuale Anteil bei 35 Prozent. Unbeliebteste Farbe von Männern ist braun. Während Männer kräftigere Töne bevorzugen, bevorzugen Frauen Pastelltöne. Unbeliebteste Farbe bei Frauen ist orange. [74] Betrachtet man die von Scott Design Inc erhobenen Daten [72], so wird die Farbe Blau Geschlechter übergreifend bevorzugt. Durch den Fakt, dass die Farbe Blau eine beruhigende und friedliche Wirkung auf die Psyche hat, wird die Anforderung auf psychischer Ebene abgedeckt, nach der ein Anwender der Anwendung mehrere Stunden mit der Anwendung arbeiten wird. Zusätzlich verursacht Blau keine all zu große Aufmerksamkeit und reduziert Stress. [73] Außerdem ist die Farbenblindheit Tritanopia (Blauschwäche) sehr selten. [75] Durch die hier für die Zielgruppe passende Eigenschaften der Farbe Blau, wird diese als eine Farbe in das Farbschema für TheClient eingehen.

Analog zur Farbe Blau ist die Farbe Grün sowohl bei Frauen als auch bei Männern die direkt nach Blau favorisierte Farbe. Gleichzeitig ist die Farbe Grün analog zur Farbe Blau eine der seltensten erwähnten Farben, die von Probanden überhaupt nicht gemocht werden. [72] Grün repräsentiert Wachstum, Geld

und Sicherheit und schont und entspannt die Augen. [76] Aus den erwähnten Gründen geht die Farbe Grün in das Farbschema der Anwendung ein.

Die Farbe Rot passt als Komplementär zu den Farben Blau und Grün. Aufgrund von Farbenblindheit einer Anzahl von Menschen, wie beispielsweise Anchromasie, wird jedoch auf die Farbe Rot verzichtet, da die Farbe Grün bereits im Farbschema ist. Weitere Konflikte würden mit der Verwendung von Rot und Grün im gleichen Farbschema auftreten. Beispielsweise bei der Verwendung der beiden Farben als 'Call-to-Action'-Elemente, wie Schaltflächen. Die Auswahl einer Schaltfläche wird schwieriger, da der Anwender nicht weiß, welcher der primäre ist. [77] Im Gegensatz dazu steigert Orange die Sauerstoffliefe-rung zum Hirn und stimuliert geistige Aktivität. Eignet sich deshalb besonders gut als 'Call-to-Action'-Element. Orange fügt Spannung hinzu, wenn man sich fühlt, als ob die Zeit kaum vergehe. Ermutigt zur Kreativität. [78]

Abbildung 4.10 zeigt ein auf drei Farben beschränktes Farbschema. Die erste Farbe ist die Hauptfarbe Blau. Die zweite Farbe Grün dient als analoge Farbe zur Hauptfarbe und wird für Überschriften und Sprungelemente verwendet. Die dritte Farbe Orange dient als komplementäre Farbe für Elemente im Design, die die besondere Aufmerksamkeit des Benutzers benötigen. Die Typografie kennt vier Varianten von Schriftgrößen. Die als 'Headline#1' bezeichnete Variante für Hauptüberschriften, die als 'Headline#2' bezeichnete Variante für Unterüberschriften, die als 'Normal' bezeichnete Variante für normalen Text und die als 'Small' bezeichnete Variante für kleine Texte. Die absoluten Schriftgrößen unterscheiden sich je nach Displaygröße des Gerätes.

Abbildung 4.10.: Farbschema der Anwendung

5. Entwurfsdesign und Implementierung

Die im vergangenen Kapitel 4 erarbeiteten Konzepte und die getroffenen Entscheidungen bilden die Grundlage für das nachfolgende Entwurfsdesign und die daran anschließende Implementierung. Durch die Komplexität der Implementierung sind die folgenden Unterkapitel auf das grundlegende Design der in Kapitel 4 beschriebenen Kernkonzepte beschränkt, die bei Bedarf durch Implementierungsdetails ergänzt werden. Die vollständige Implementierung von TheClient findet sich als Quellcodeprojekt auf beiliegender DVD.

5.1. Testumgebung

Die Technologie UA agiert innerhalb eines Client-Server Architekturkontextes. Die unterliegende Kommunikation verläuft dabei wie beschrieben durch TCP. Die Kommunikation von verteilten Systemen über Prozessgrenzen hinweg in möglicherweise unterschiedlichen Netzwerken erfordert immer auch eine Konfiguration aller beteiligten physikalischen und virtuellen Komponenten, die an der entsprechenden Kommunikation beteiligt sind. Eine signierte und verschlüsselte Kommunikation, welche in UA Anwendungen gemäß Standard ermöglicht werden muss, erhöht den Konfigurationsaufwand zusätzlich.

Um einen Discovery zu ermöglichen, von dem die IP-Adresse beziehungsweise der Hostname des Discovery-Servers unbekannt ist, muss über den Ethernet-Port *4840* ein Datenaustausch zwischen Sender und Discovery Server ermöglicht werden. Um den TCP-Datenaustausch auch über Rechnergrenzen, Firewall-Grenzen und Betriebssystemgrenzen hinweg zu ermöglichen muss

der Port *4840* für alle Kommunikationskomponenten auf der Kommunikationsstrecke unblockiert sein. [1, 29, 32, 39] Zur Auslieferung des Standardinformationsmodells und zur Auslieferung des *OPC UA Informationsmodell für FDT* wurden zweierlei UA Server in die Testumgebung integriert. Zur Auslieferung des Standardinformationsmodells mit entsprechenden Beispieldaten wurde der Referenz Server [79] der verwendet. Dieser ist nach dem Start des Servers standardmäßig aus Sicht der lokalen Workstation über die Adresse *opc.tcp://localhost:62541/Quickstarts/ReferenceServer* erreichbar.

Der UA Standard definiert zwei zu unterstützende Protokolle. Einmal das Binärprotokoll *opc.tcp*, mit welchem die hier beschriebenen Serveradressen ausgezeichnet sind, und zum anderen das *HTTP*-Protokoll für Webdienste. Die hier genannten Adressen können somit genauso über das *HTTP*-Protokoll anstatt dem hier exemplarisch verwendeten *opc.tcp*-Protokoll erreicht werden. [80]

Zur Repräsentation *des OPC UA Informationsmodell für FDT* wird der UA Server des Unternehmens Unified Automation [81] in einer Testversion verwendet. Hierbei verwaltet eine Komponente im UA Server die Zuordnung von FDT Daten zu Datenknoten innerhalb des *OPC UA Informationsmodell für FDT*. Die Zuordnungssoftware benötigt hierzu die Gerätetopologie der beteiligten unterliegenden physikalischen Geräte, die in FDT durch die DTMs repräsentiert werden. Von der Zuordnungssoftware werden derzeit ausschließlich DTMs unterstützt, welche auf der Version zwei von FDT basieren.

Abbildung 5.1 zeigt die FDT-Topologie, die für den Datenabruf, die Datenvisualisierung und Datenmodifizierung von FDT Informationen in TheClient verwendet wurde. Hierzu wurden FDT2 DTMs verwendet, zu denen beim Unternehmen M&M Software GmbH keine physikalischen Geräte verfügbar waren. Aus diesem Grund wurde anstatt den Geräten eine Simulationssoftware eingesetzt, die reale physische Geräte imitieren kann.

Der UA-Server, der die Zuordnung von FDT Daten zum *OPC UA Informationsmodell für FDT* repräsentiert, kann standardmäßig über den Port *48030* unter der Adresse *opc.tcp://localhost:48030* kontaktiert werden. Eingehende und ausgehende Kommunikation über diesen Port dürfen für eine erfolgreiche

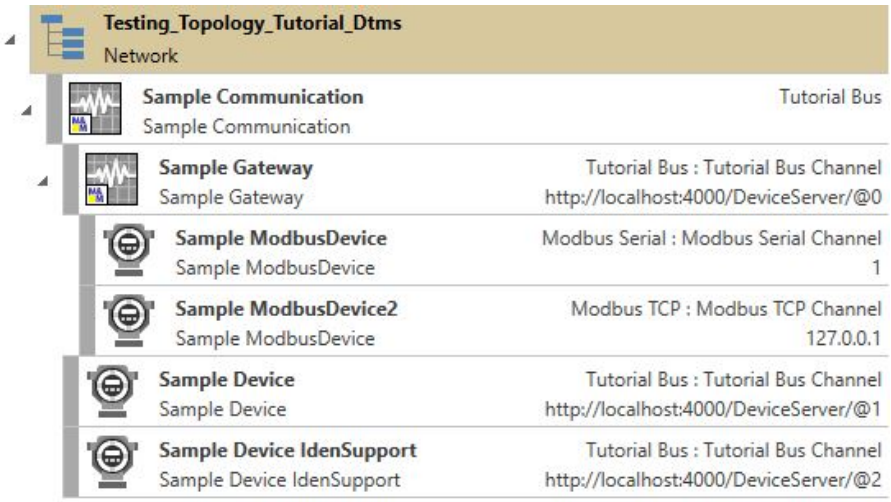

Abbildung 5.1.: Field Device Tool Topologie, deren beinhaltete Device Type Manager simulierte Gerätedaten für das OPC UA Informationsmodell bereitstellen

Kommunikation nicht blockiert werden. Gleiches gilt für den Port *62541* des UA Servers, der das Standardinformationsmodell ausliefert.

5.2. Architektur

Die Anwendungsarchitektur in Abbildung 5.2 zeigt die grundlegenden Anwendungsübergreifenden Eigenschaften des Lösungskonzepts für die Implementierung von TheClient. [82, 83]

Die grundlegende Anwendungsarchitektur für TheClient basiert auf dem Architekturentwurfsmuster Model View ViewModel (MVVM). Dieses Muster erlaubt die strikte Trennung zwischen Visualisierung und Geschäftslogik, die bei Betrachtung der Unabhängigkeit der UA Technologie ein weiterer konsequenter Schritt zur vollständigen Unabhängigkeit aller Komponenten bedeutet. Mit der Trennung des Architekturentwurfsmusters wird zusätzlich eine strukturierte Programmstruktur und die einfache Erweiterungsmöglichkeit für alle in Kapitel 4 beschriebenen Konzepte und Folgekonzepte ermöglicht.

Die Anwendungsarchitektur von TheClient, die in Abbildung 5.2 illustriert ist,

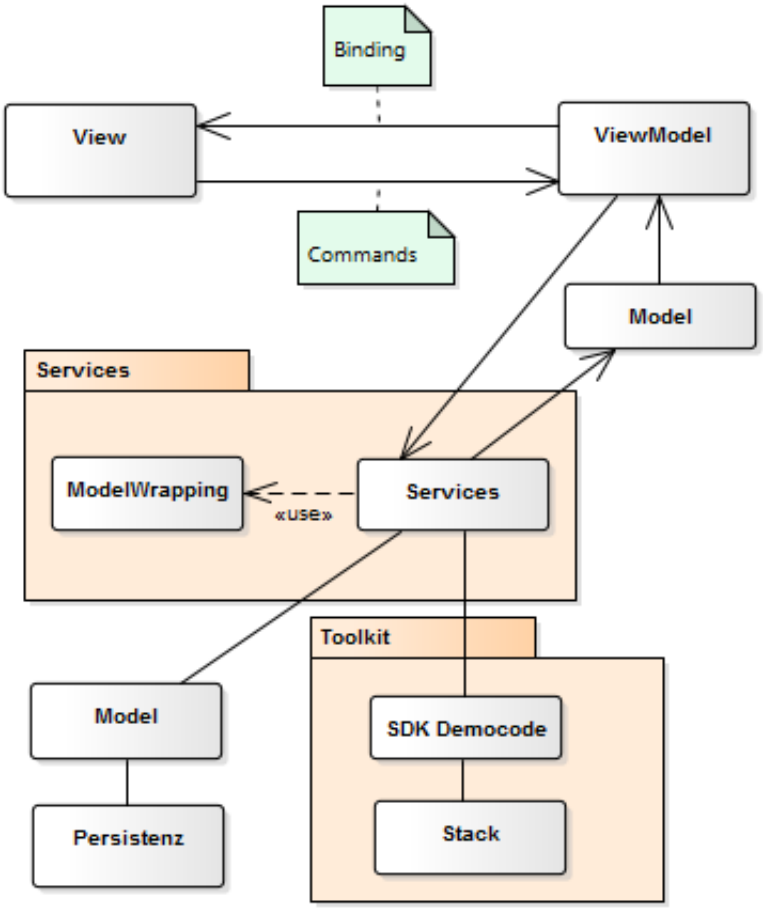

Abbildung 5.2.: Grundlegender Aufbau der Anwendung TheClient nach dem Architekturentwurfsprinzip Model-View-ViewModel

zeigt die verschiedenen Komponenten - Model, View und ViewModel - gemäß dem MVVM-Architekturentwurfsmuster. Die Service-Komponente realisiert die Geschäftslogik unter Verwendung der in der Abbildung repräsentierten Persistenz- und Toolkitkomponente. Der Austausch zwischen Service-Komponente und den unterliegenden Komponenten wird über vorher definierte Datenmodelle gemäß dem MVVM-Entwurfsmuster realisiert. Die View Komponente ist für die Definition der Visualisierungsart verantwortlich. Die zu visualisierenden Informationen werden der View-Komponente dabei von der View-Model Komponente bereitgestellt. Die Trennung der eigentlichen Information und der Visualisierung der Information ist für die prototypische Implementierung von TheClient von besonderer Bedeutung, gemäß der Forschungsfra-

ge, welche eine Unabhängigkeit der Information erwähnt. Die Wichtigkeit zeigt sich im Anwendungsdesign für die grafische Benutzeroberfläche in Kapitel 5.6. Die Trennung von Information und Informationsvisualisierung ermöglicht durch das Entwurfsmuster MVVM, die Implementierung mehrerer Views, welche die gleichen Informationen über dasselbe ViewModel konsumieren, die gleichen Daten aber auf unterschiedliche Art visualisieren. Zur Schaffung der größtmöglichen losen Kopplung wird die Kommunikation zwischen den Komponenten View und ViewModel über Bindungen und Kommandos realisiert. Bindungen ermöglichen Eigenschaften einzelner ViewModels an die Struktur einzelner Views zu binden. Die zu bindenden Eigenschaften werden dabei in der View definiert. Dem entgegen ermöglichen Kommandos der View, bestimmte Operationen aufzurufen, die ein einzelnes ViewModel bereitstellt. Die Nutzung sogenannter Datenvorlagen sind im Zusammenhang der Implementierung einzelner Views von besonderer Bedeutung. Die Datenvorlagen werden dabei direkt in die jeweilige View integriert. In der Theorie dient die ViewModel-Komponente als Verbindungsstück zwischen den Komponenten View und Model. Das Model repräsentiert dabei die Daten, die von einer View visualisiert werden soll. Hierbei können die Daten statisch im Model vorliegen oder aber dynamisch über beispielsweise Webdienste und Datenbanken konsumiert werden. [5, 84]

5.2.1. Datenkonsum

Bei der Implementierung von TheClient wird der Datenkonsum über die Abstraktionskomponente Service realisiert. Der Datenkonsum wird dabei durch ein entsprechendes ViewModel durch Aufruf passender Dienste innerhalb der Service-Komponente initiiert. Nach der Initiierung konsumiert der entsprechende Service die Daten von den unterliegenden Schichten Persistenz und Toolkit, bereitet die konsumierten Daten auf und legt diese Daten in entsprechenden Modelklassen ab. Das entsprechend mit Daten befüllte Modell wird an das anfordernde ViewModel übergeben, von welchem aus die Daten des Modells über Bindungen an die entsprechenden View Instanzen weitergereicht werden. Die Daten werden dabei an alle View Instanzen weitergereicht, welche zuvor

eine Bindung an die entsprechenden Eigenschaften der ViewModel Instanz definiert haben.

Die in der Anwendungsarchitektur abgebildete Persistenz wird durch zwei Konzepte realisiert: Relationale Datenbank und Schlüssel/Wert Speicher. Die relationale Datenbank wird in TheClient durch SQLite mit entsprechender objektrelationaler Abbildung realisiert, zur Speicherung größerer zusammenhängender Daten. Die objektrelationale Abbildung soll in TheClient die Speicherung vereinfachen, in dem die sowieso bestehenden Modelle direkt an den objektrelationalen Abbilder gehen und von dort den Weg zur Datenbank finden. Bei den zu persistierenden Daten kann es sich exemplarisch um ein Datenabonnement gemäß der definierten Minimalanforderungen handeln, welcher wiederum die zu beobachtenden Datenknotenidentifikationsnummern enthält. Alle notwendigen Daten für das Datenabonnement werden zur Laufzeit als Instanz in einem Modellobjekt gespeichert und können zum Ende der Laufzeit entsprechend persistiert werden. Der Schlüssel/Wert Speicher wird entgegengesetzt zur relationale Datenbank für die Speicherung nicht zusammenhängender, kleiner Daten genutzt. Alle Plattformen die beinhalten ein eigenes Konzept für Schlüssel/Wert Speicher, die in Xamarin adressiert werden können. Bei der Implementierung von TheClient haben sich diese beiden Persistenzkonzepte bewährt, da der Schlüssel/Wert Speicher bei Erstimplementierung nicht geeignet war für die Speicherung zusammenhängender Daten die in TheClient anfallen. In Analogie dazu eignet sich SQLite aufgrund des Setups einer relationalen Tablle für jeden weiteren Datensatz, nicht für die Speicherung kurzfristiger Werte die in TheClient ebenfalls anfallen, wie beispielsweise Fortsetzungspunkte beim Laden von Datenknoten aus dem UA Modell.

Die als Toolkit bezeichnete Komponente innerhalb der Anwendungsarchitektur von TheClient wird durch zwei Unterkomponenten repräsentiert: SDK Democode und UA Stack. Während es sich beim UA Stack um den portierten UA Stack der OPC Foundation handelt, enthält die Unterkomponente SDK Democode Teile des zu verwendenden Beispielquellcodes der OPC Foundation zu Abstraktion des UA Stacks, welches in der Theorie durch die SDK-Schicht realisiert wird. Durch die nicht Existenz eines SDKs, wird das SDK durch den Beispielquellcode der OPC Foundation innerhalb der Toolkit-Komponente

und durch Implementierungsdetails in den Diensten innerhalb der Service-Komponente repräsentiert. Die Kommunikation zwischen den Diensten und den darunterliegenden Schichten Persistenz und der Toolkit-Komponente mit dessen Unterkomponenten wird ebenfalls durch Methodenaufrufe realisiert. Die Datenantwort der aufgerufenen Methoden erfolgt über Modellobjekte, die allerdings womöglich aus Gründen der Datenabstraktion oder Datensicherheit nicht direkt an die jeweilige ViewModel-Instanz weitergereicht werden können, sondern über die Service-Komponente entsprechend einer anderen Modellstruktur zugeordnet werden. In Abbildung 5.2 wird dieser Sachverhalt durch `ModelWrapping` repräsentiert. So kann das Toolkit Modellobjekte liefern, welche für die prototypische Implementierung nicht benötigte Daten enthält oder die aufgrund der Lese/Schreibe-Berechtigung des Datenknotens nicht in der grafischen Benutzeroberfläche angezeigt werden sollen. Alternativ hätten die Modellobjekte des SDK Democodes angepasst werden können. Aufgrund der in vorherigen Kapiteln aufgestellten Regel, möglichst wenig Änderungen am originalen Quellcode vorzunehmen, wurde dies unterlassen und durch die Klasse `ModelWrapping` realisiert.

5.2.2. Datenvisualisierung

Zur Realisierung einer vollständigen losen Kopplung der Komponenten View und ViewModel, werden die View-Klassen formal mit Extensible Application Markup Language (XAML) beschrieben. Zusätzlich wurde zur Zielerreichung in der prototypischen Implementierung der Infrastrukturquellcode zur Verwaltung von Bindung und Kommandierung auf ein Minimum reduziert. Der zu abstrahierende Infrastrukturquellcode würde regulär wie in Abbildung 5.3 aussehen.

Der abgebildete Quellcodeausschnitt ist für jede View notwendig, um sich mit entsprechendem ViewModel und damit, mit einem Datenlieferanten zu verbinden. Jede View-Instanz erbt dabei von der Klasse `ContentPage`, die durch *Xamarin.Forms* bereitgestellt wird, zur Repräsentation einer unabhängigen Seite in der grafischen Benutzeroberfläche. Durch die View-Instanz wird die ent-

```
1  public partial class ViewOne : ContentPage
2  {
3          ViewOneViewModel _viewModel;
4          public ViewOne()
5          {
6                  InitializeComponent();
7                  _viewModel = new ViewOneViewModel();
8                  BindingContext = _viewModel;
9          }
10 }
```

Abbildung 5.3.: Regulärer Infrastrukturquellcode zur Kopplung von View und ViewModel zum Ziel des Datenkonsums

sprechend zugehörige ViewModel-Instanz erzeugt und entsprechende Bindungen definiert, von welcher die View-Instanz Daten konsumieren möchte.

Die Abbildung 5.4 zeigt die für TheClient angepasste Detailarchitektur zur Erzeugung und Erhaltung der strikten Trennung zwischen View zur Visualisierung und der ViewModel als Bindeglied zur Geschäftslogik. Hierbei wird die Infrastrukturerzeugung aus den View-Klassen ausgelagert.

Die Erzeugung und Beibehaltung der Infrastruktur für die Komponenten View und ViewModel wird, wie Abbildung 5.4 zeigt, durch die Utils-Komponente realisiert. Auf unterer Ebene findet sich die View-Komponente, die durch den XAML-Markupquellcode die Benutzeroberfläche formal beschreibt und durch das in der Abbildung aufgezeigten XAML-Backend unterstützt wird. Der Infrastrukturquellcode wird traditionell durch das XAML-Backend repräsentiert und wird durch die Utils-Komponente abstrahiert. Das XAML-Backend repräsentiert dennoch weiterhin entsprechende *Eventhandler*, denen bestimmten grafischen Elementen zugeordnet sind. Der Infrastrukturquellcode wird in die Klasse `ViewBase` ausgelagert. Abbildung 5.5 zeigt die Verknüpfung der View mit dem ausgelagerten Infrastrukturquellcode in `ViewBase`.

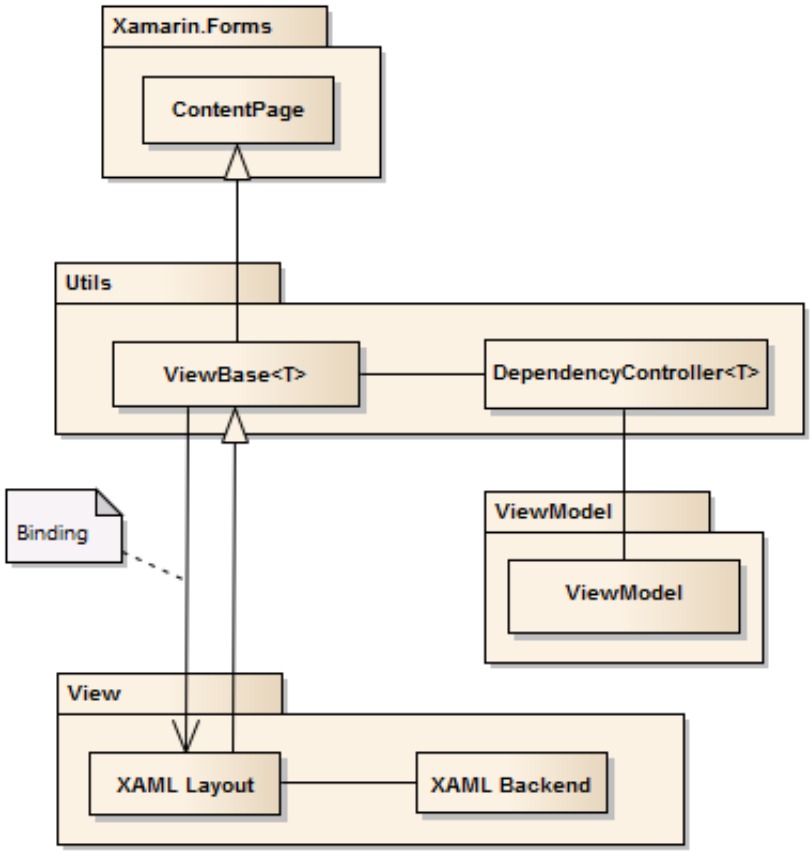

Abbildung 5.4.: Architektonische Infrastruktur zur strikten Trennung zwischen der View zur Visualisierung und dem ViewModel als Bindeglied zur Geschäftslogik

```
1  <utils:ViewBase xmlns="http://xamarin.com/schemas/2014/
       ↪ forms"
2           xmlns:x="http://schemas.microsoft.com/winfx/2009/
               ↪ xaml"
3           x:Class="TheClient.View.ViewOne"
4           x:TypeArguments="viewModel:ViewOneViewModel"
5           xmlns:viewModel="clr-namespace:TheClient.
               ↪ ViewModel;assembly=TheClient"
6           xmlns:utils="clr-namespace:TheClient.Utils;
               ↪ assembly=TheClient"
7           Title="{Binding␣PageTitle}">
8
9                ...
10
11 </utils:ViewBase>
```

Abbildung 5.5.: Verknüpfung einer View mit dem in die Klasse ViewBase ausgelagerten Infrastrukturquellcode

Die Verknüpfung wird durch Notieren des entsprechenden Datenlieferanten realisiert. In der Abbildung 5.5 wird der Datenlieferant durch `ViewOneViewModel` repräsentiert. Die `ViewBase`-Instanz instanziiert den entsprechenden Datenlieferanten `ViewOneViewModel` durch die Nutzung der Abhängigkeitsverwaltung, die in der Abbildung 5.2 als `DependencyController` benannt wurde. Die Bindung von View und ViewModel wird bei der Implementierung von TheClient damit ebenfalls durch die `ViewBase` realisiert, anstatt direkt durch die ViewModel.

5.2.3. Abhängigkeitsverwaltung

Bei der Abhängigkeitsverwaltung handelt es sich um ein, durch die vollständige Architektur durchzogenes Konzept um Abhängigkeiten zwischen Instanzen zu kontrollieren. Es wird unterschieden zwischen zwei Arten von Abhängigkeitsverwaltungen, die dezentrale und zentrale. Bei der dezentralen Abhängigkeitsverwaltung, welche exemplarisch in Abbildung 5.3 illustriert wurde, werden Abhängigkeiten zwischen Instanzen direkt durch die Instanzen selbst verwaltet. Es wird ersichtlich, wie die Instanz `ViewOne` aus der Komponente View eine Instanz der Klasse `ViewOneViewModel` aus der Komponente ViewModel erzeugt. Die zentrale Abhängigkeitsverwaltung hingegen wird durch eine zentrale Instanz realisiert, die nachfolgend beschrieben wird. Zur Verknüpfung von gemeinsamen Quellcode mit dem spezifisch ausgelagerten Quellcode einer Plattform wird aus den in Kapitel 4.2 genannten Gründen die zentrale Abhängigkeitsverwaltung durch Dependency Injection bevorzugt und für die Implementierung von TheClient genutzt. In Abbildung 5.6 wird die in TheClient zu realisierende, zentrale Abhängigkeitsverwaltung grafisch dargestellt.

Die in Kapitel 4.3.4 beschriebene Modifizierungsstrategie erläutert die Auslagerung von Quellcodeteilen durch Kapselung von Methoden. Um durch die Abhängigkeitsverwaltung zu realisierende Abhängigkeit zwischen gemeinsamen und spezifischen Quellcode herzustellen, wird der Methodenkopf, der den ausgelagerten Quellcode repräsentiert im Interface hinterlegt. Alle von TheClient genutzten Plattformprojekte implementieren das entsprechende Interface und müssen zugleich gemäß dem objektorientierten Programmierparadigma die

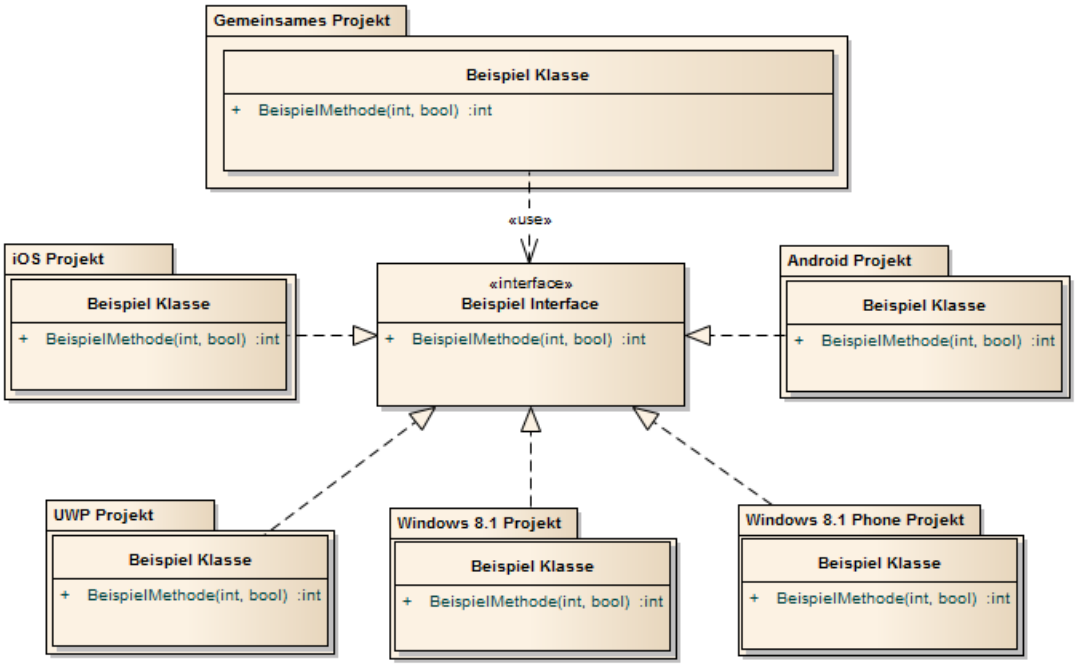

Abbildung 5.6.: Abhängigkeitsverwaltung zwischen gemeinsamen und spezifischen Quellcode der Plattformen

Implementierung der Methodenfunktionalität für die entsprechende Plattform realisieren. Aus der Abbildung geht hervor, dass der Zugriff des gemeinsamen Projekts auf in spezifisch ausgelagerte Funktionalitäten, durch das Interface realisiert wird. Bei der Kompilierung durch Xamarin werden entsprechend dem Kompilierungsverfahren von Xamarin, der gemeinsame Quellcode und der plattformspezifische Quellcode des zu kompilierenden Projekts ausgewählt. Dadurch ergeben sich keine Konflikte bei der Auswahl der Implementierung, beim Zugriff über das Interface, da lediglich die Implementierung des kompilierten Plattformprojekts zur Verfügung steht. Wird TheClient beispielsweise unter UWP ausgeführt, so wird lediglich der Quellcode des gemeinsamen Projekts und der Quellcodes des UWP Projekts zur Laufzeit verfügbar. Wird im gemeinsamen Projekt über das Interface auf die `BeispielMethode` zugegriffen, wird unweigerlich die `BeispielMethode` innerhalb des UWP Projekts ausgeführt, da lediglich die spezifische Implementierung von UWP zur Verfügung steht.

Zur zentralen Verwaltung der Abhängigkeiten von TheClient, die in der Theo-

rie beschrieben wurden, wird für den praktische Verwendung in TheClient das NuGet-Paket Autofac [85] verwendet, das die Registrierung der Abhängigkeiten gemäß des beschriebenen Konzepts vereinfachen soll. Diese Drittkomponente erlaubt die Abstraktion von Registrierung der Abhängigkeit und Zugriff auf durch die Abhängigkeit gebundenen Komponenten. Autofac basiert auf sogenannten Containern, die einen Bereich für zu verwaltende Abhängigkeiten repräsentieren.

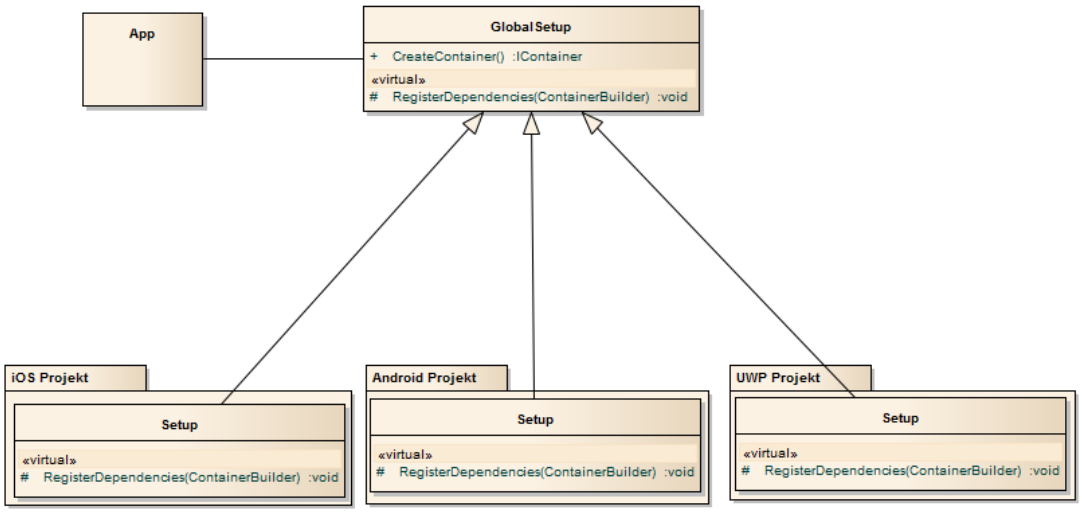

Abbildung 5.7.: Registrierung von Abhängigkeiten

Die Registrierung von Abhängigkeiten in TheClient wird, wie Abbildung 5.7 zeigt, durch Setup-Klassen realisiert. Hierbei werden alle Abhängigkeiten, die unmittelbar das gemeinsame Projekt betreffen, über die Klasse `GlobalSetup` durch die Methode `RegisterDependencies` registriert. Die Registrierung von Abhängigkeiten speziell in die Plattformen ausgelagerten Quellcodes, werden durch die Plattformen selbst durch spezielle Setup-Klassen realisiert. Die Setup-Klassen erben dabei von der Klasse `GlobalSetup` und überschreiben die Methode `RegisterDependencies`.

In Abbildung 5.8 wird exemplarisch die Implementierung der Methode `RegisterDependencies` durch das plattformspezifische Projekt Android realisiert. Die Abbildung zeigt gleichzeitig die Rücksichtnahme auf den vorherigen Aufruf der Basis Methode in der Klasse `GlobalSetup`, damit neben den plattformspezifischen Abhängigkeiten weiterhin die plattformübergreifenden Abhängigkeiten

zur Verwendung registriert werden.

```
1 internal class Setup : GlobalSetup
2 {
3         protected override void RegisterDependencies(
              ↪ ContainerBuilder cb)
4         {
5                 base.RegisterDependencies(cb);
6                 cb.RegisterType<CryptoServiceDroid>().As<
                      ↪ ICryptoService>();
7         }
8 }
```

Abbildung 5.8.: Exemplarische Implementierung der Methode RegisterDependencies durch das plattformspezifische Projekt Android

Auf die registrierten Abhängigkeiten im durch Autofac erzeugten Container muss in der vollständigen Anwendung *TheClient* zugegriffen werden können. Zu diesem Zweck wird die statische Klassendeklaration `DependencyController` angelegt, die eine Referenz zum Container und damit zu den Abhängigkeiten besitzt.

5.3. Portierung des UA Stacks

Wie in den vergangenen Kapiteln erläutert bildet die Portierung des UA Stacks nach Xamarin zur plattformübergreifenden Entwicklung, die Grundlage zum Datenabruf und der Datenmodifizierung. Erst durch eine lauffähige Portierung kann der Stack entsprechend durch eine SDK-Schicht abstrahiert werden und die Möglichkeit zur Visualisierung der Daten besteht.

Wie bereits aus dem Abschnitt 4.3.1 hervorgeht, ist die einzige Möglichkeit die Integration des Stacks der von der OPC Foundation angeboten wird. Dieser wird durch die in Abschnitt 4.3.3 beschriebene Integrationsstrategie in Xamarin integriert. Bei der Portierung erwies sich die gewählte Integrationsstrategie allerdings als nicht zielführend. Der Grund hierfür war die zu hohe Anzahl an

Fehler verursachenden Quellcodestellen, die mit der gewählten Integrationsstrategie gemäß den in Kapitel 4.3.3 aufgeführten Nachteilen der Strategie, zu aufwendig und komplex wurde. Die Portierung des Stacks erfolgte bei TheClient durch die Alternativstrategie. Hierzu wurde in einem ersten Schritt ein Projekt innerhalb der zu verwendenden Entwicklungsumgebung Visual Studio mit Xamarin, nach der Vorlage *Xamarin Forms Shared Project* angelegt. Anschließend wurden die Dateien des UA Stacks in das gemeinsame Quellcodeprojekt der Anwendungslösung transferiert, unter Beachtung der Beibehaltung von Eigenschaften in Relation des Originalprojekts des UA Stacks. Als Eigenschaften wurden hier die zwei folgenden Punkte betrachtet: a) Ist die Datei in der Beispielanwendung inkludiert? b) Welche Build Action hat die Datei in der Beispielanwendung?

Die Quellcodeanalyse aus Kapitel 4.3.2 zeigt, dass innerhalb der Xamarin Entwicklungsumgebung für keine Plattform eine Lauffähigkeit bei Integration des UA Stacks besteht. Grund hierfür sind die in den vergangenen Unterkapiteln erläuterten, fehlenden Typen des .NET-Rahmenwerks, die im UA Stack verwendet werden. Bei der Portierung wurden nun die durch die Fehleranalyse erkannten Quellcodestellen umgeschrieben, sodass diese keine Typen verwenden, die von der Plattform nicht unterstützt werden. Ziel hierbei war die Umschreibung, so dass diese Quellcodestellen nach der Umschreibung auf allen Plattformen lauffähig sind. War dies nicht möglich, wurde der Quellcode in das spezifische Projekt verlagert und dort durch Umschreibung zur Lauffähigkeit gebracht. Aufgrund der Masse an Fehler verursachenden Quellcodestellen ist dieser Prozess langwierig. Im Verlaufe dieser Modifizierung veröffentlichte die OPC Foundation einen alternativen UA Stack, welcher für die Verwendung speziell auf UWP Geräten entwickelt wurde. Durch die Analyse des alternativen Stacks in Kapitel 4.3.2 zeigt sich, dass dieser aufgrund geringerer Fehlercodestellen im Gegensatz zum originalen Stack der OPC Foundation, als geeigneter einzustufen ist. Obwohl in einem zeitintensiven Prozess bereits massive Teile vom Quellcode des Original Stacks mit dem Ziel der Lauffähigkeit modifiziert wurden, entschied der Autor dieser Arbeit die Modifizierung einzustellen und den Wechsel zur Integration und Modifikation des alternativen UWP Stacks anzustreben.

Bei der Integration des UWP Stack wurde ebenfalls nach der alternativen Integrationsstrategie vorgegangen. Die Modifizierung erfolgte nach der in Kapitel 4.3.4 beschriebenen Modifizierungsstrategie unter Verwendung der Architekturprinzipien aus Kapitel 5.2. Für einige der nicht nutzbaren Typen des .NET-Rahmenwerks konnten gesonderte Pakete referenziert werden, welche Teile der fehlenden Typen nutzbar machten. Exemplarisch sei hier für Android und iOS das Paket `Serialization` erwähnt, welches die .NET-Rahmenwerk-Teilmenge `System.Runtime.Serialization` für Android und iOS verfügbar macht, während für UWP die Teilmenge standardmäßig zur Verfügung steht. Aufgrund der dennoch sehr hohen Anzahl an Fehler verursachenden Quellcodestellen wurde die Portierung zu Beginn auf die Plattformen Android, iOS und UWP beschränkt, da dies die auf dem Markt meist vertretenen Plattformen sind. Aufgrund des hohen Zeitbedarfs der Portierung wurde diese im Laufe der Forschungsarbeit eingestellt mit der Erkenntnis, dass einer Portierung mit genügend Zeitbedarf realisierbar ist. Die Lauffähigkeit des Stacks innerhalb der Xamarin Entwicklungsumgebung wurde mit der Lauffähigkeit des Stacks unter UWP nachgewiesen.

Nach der Portierung des UA Stack wurde gemäß der in Unterkapitel 5.2 beschriebenen Architektur, das SDK, die Persistenz und die zur Abstraktion genutzten Services entwickelt, die in nachfolgenden Unterkapiteln erläutert werden.

5.4. Services und SDK

Die Service Komponente erhält in TheClient eine besondere Bedeutung. Die Schicht nutzt die Persistenz Komponente und die Toolkit Komponente. Abbildung 5.9 zeigt diesen Sachverhalt. Wie in der Abbildung zu sehen enthält die Toolkit Komponente den wie in Kapitel 5.3 beschriebene, portierten UA Stack. Das auf dem UA Stack aufbauende SDK wurde in TheClient in zweierlei Teile getrennt. Zum einen wurde, wo es Sinn machte, der Democode der Beispielanwendung des Original UA Stacks genutzt. Die fehlenden und ergänzenden

Teile hierzu wurden entsprechend in den einzelnen Service Klassen hinterlegt.

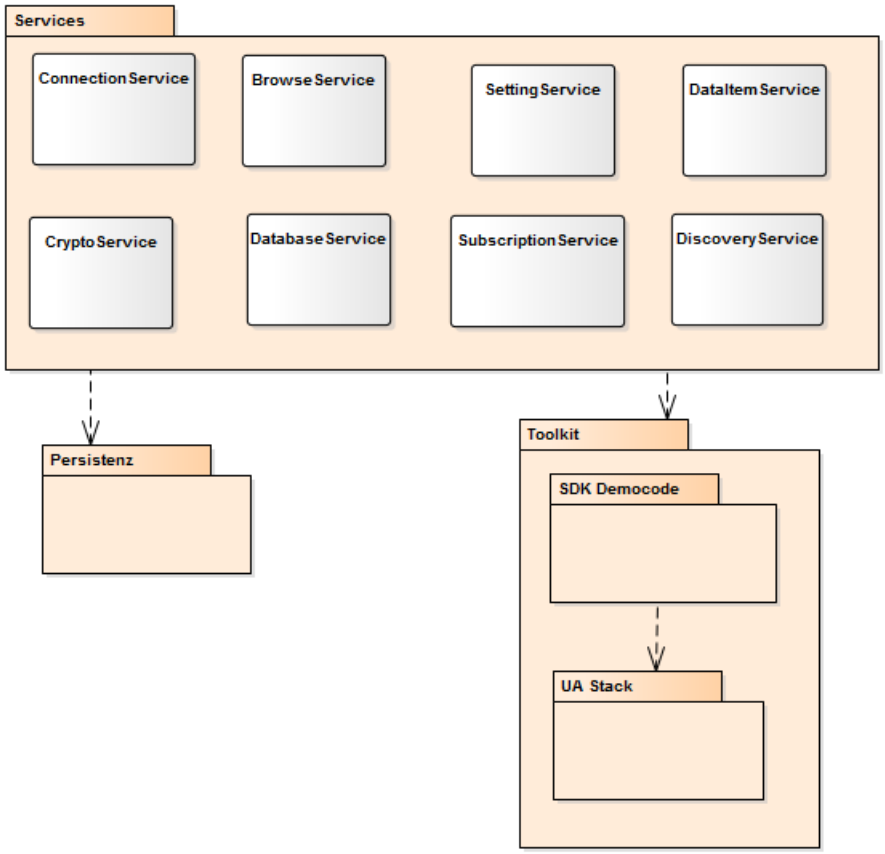

Abbildung 5.9.: Abstraktion der Persistenz- und Toolkitkomponente durch die Service-Komponente

Für TheClient wurden acht Service Klassen identifiziert, diese können von einem ViewModel der ViewModel Komponente adressiert werden. Das entsprechende ViewModel erhält die Daten des Dienstaufrufs entsprechend als Modelobjekt zurück.

5.5. Persistenz

Neben dem Toolkit steht der Peristenz Komponente ebenfalls eine besondere Funktion in TheClient zu. Die Beispielanwendung des UA Stacks, verwen-

det zur Grundkonfiguration Extensible Markup Language (XML) Dateien. Für TheClient wurde die Beibehaltung dieser XML Konfigurationsdateien angestrebt, damit der SDK Democode ohne größere Änderungen in der Toolkit Komponente verwendet werden kann. Einige dieser Konfigurationen, welche anwendungsweit in TheClient verwendet werden sollten, wurde dabei aus der XML Konfiguration in die Peristenz Komponente ausgelagert. Weiterhin dient die Komponente, neben der Auslagerung von Konfigurationseigenschaften zur Persistierung anfallender Zustandsinformationen in TheClient.

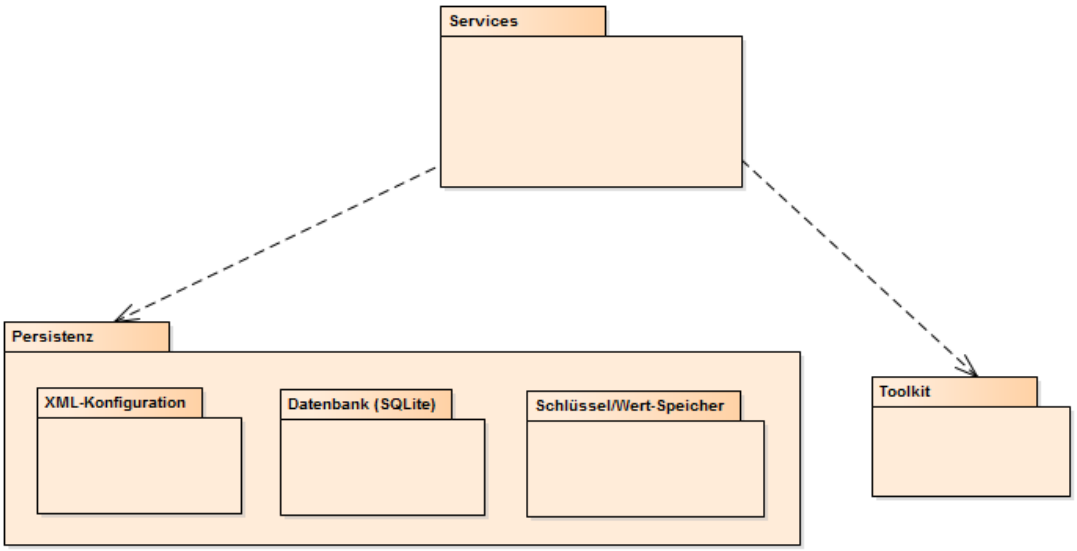

Abbildung 5.10.: Dreiteilige Peristenzlösung

Abbildung 5.10 zeigt, dass die Peristenz Lösung aus drei Teilen besteht: XML-Konfiguration, Datenbank sowie Schlüssel/Wert Speicher. Zur strukturierten Datenhaltung beliebig komplexer und in der Regel zusammenhängender Daten, wird mittels einer relationalen Datenbank realisiert. Es wurden verschiedene Datenbankbibliotheken analysiert, mit dem Ergebnis dass die Programmbibliothek SQLite, zur Einbindung eines relationalen Datenbanksystems besonders geeignet ist.

Die besondere Eignung ergibt sich durch die Unterstützung von SQLite für Android, iOS und UWP. [86] Die Einbindung von SQLite erfolgte in TheClient zusätzlich mit der Verwendung eines OR-Mappers, um Objekte direkt an SQLite zu übergeben, welche daraus automatisiert die Umwandlung in das relationale

Schema anstößt. Werden Daten aus der Datenbank gelesen, wird in Analogie die als relationales Schema zur Verfügung stehenden Daten in Objekte umgewandelt, zur in der Anwendung weiteren Verarbeitung. Diese Abbildung wird durch die SQLite Erweiterung SQLite-Net Extensions [87] realisiert. Die Modellobjekte und Beziehungen der Modelle gemäß dem OR-Mapper wurde in TheClient durch das in Abbildung 5.11 aufzeigende Entity-Relationship (ER) Modell realisiert.

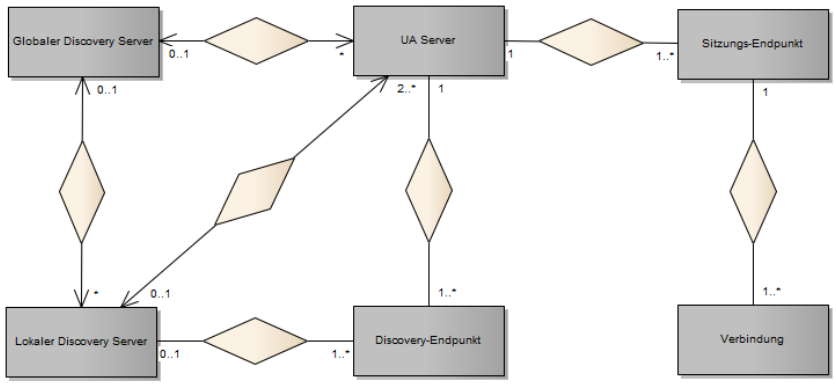

Abbildung 5.11.: Entity-Relationship-Modell für TheClient

Die Schlüssel/Wert-Speicher werden durch die Schlüssel/Wert Speicher der einzelnen Plattformen realisiert.

5.6. Benutzeroberflächengenerierung

Je nach Typ der Daten aus dem UA Informationsmodell muss es passende Moleküle geben, die auf der spezifischen Vorlage entsprechend platziert werden können. Für TheClient werden Moleküle und Atome als Komponente zusammengefasst, das heißt eine Komponente beinhaltet das Molekül und seine Atome. Dies erzeugt möglicherweise Redundanzen, hält die Programmlogik allerdings schlank und ist für eine prototypische Anwendung zur Beantwortung der Forschungsfrage ausreichend. Redundanzen entstehen in diesem Kontext dadurch, dass in einer Komponente Design-Elemente, wie beispielsweise eine Schaltfläche angelegt werden, welche bereits in einer anderen Komponente angelegt wurden. Bei der strikten Trennung des Konzepts von Molekül

und Atom hingegen würde ein Molekül per Referenz nur auf die Atome verweisen die benötigt werden. Durch dieses Prinzip würde eine Schaltfläche als Atom angelegt werden und kann von beliebig vielen Molekülen referenziert werden. Im Rahmen der prototypischen Realisierung ist die Zusammenfassung von Molekülen und Atomen als Komponenten hinzunehmen, auch wenn hierdurch Redundanzen entstehen. Für TheClient sollen einige wenige Komponenten implementiert werden, welche die unterschiedlichen Konzepte zur Bereitstellung der Daten im UA Informationsmodell repräsentieren und so die Frage der Visualisierung beantworten.

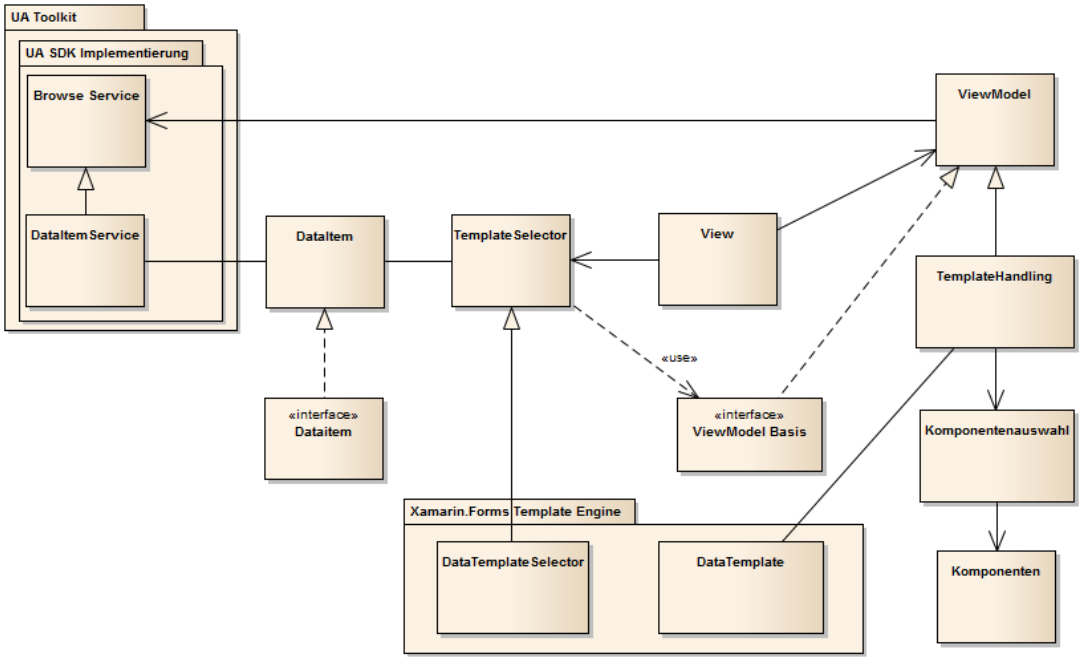

Abbildung 5.12.: Architektonischer Aufbau der Benutzeroberflächenkomponentengenerierung

Abbildung 5.12 zeigt den architektonischen Aufbau des automatisierten Aufbaus der grafischen Benutzeroberfläche. Dabei wird der Automatismus in das MVVM Grundgerüst der Architektur integriert, wie in Kapitel 5.2 beschrieben. In der Komponenten Klasse (Abbildung rechts unten) sind die grafischen Komponenten in Xamarin.Forms als XAML definiert, die durch die Steuerungsklasse `Komponentenauswahl` bei für die grafische Benutzeroberfläche passenden Eigenschaften zur Einbindung in diese weitergereicht werden. Die Aus-

wahl der Komponenten durch die `Komponentenauswahl` Klasse erfolgt hierbei durch eine Entscheidungslogik, welche die Entscheidungen auf Basis von vorher definierten Regeln trifft. Die `Komponentenauswahl` Klasse repräsentiert somit eine sehr vereinfachte Form einer Regelmaschine wie beispielsweise Drools [88]. Die Entscheidungslogik und damit verbundenen Regeln ist in Kapitel 4.4.3 genauer spezifiziert. Die `Komponentenauswahl` Klasse wird von der Klasse `TemplateHandling` angestoßen, welche die Komponente als `DataTemplate` umwandelt. `DataTemplate` ist hierbei eine Klasse aus Xamarin.Forms die mit dem Konzept der Komponente verglichen werden kann. Die Umwandlung der auf Xamarin.Forms in XAML basierten Komponenten nach DataTemplates ist notwendig, um die Komponente innerhalb von anderer Xamarin.Forms Konzepten, wie beispielsweise der im nachfolgenden beschriebenen ListView zu nutzen. Neben der Umwandlung der reinen Komponente in das Xamarin.Forms Konzept DataTemplate, enthält die Klasse außerdem einige Verwaltungsfunktionen, die unter der Nutzung der DataTemplate Klasse von Xamarin.Forms wie im folgenden erläutert, notwendig sind.

Xamarin.Forms schreibt zur Vermeidung von Speicherlecks (memory leaks) vor, dass für dasselbe DataTemplate und damit dieselbe Komponente keine erneute Instanz erzeugt werden darf. Zudem muss für die gleiche Datenkombination innerhalb einer Komponente immer das gleiche DataTemplate zurückgegeben werden. Die `TemplateHandling` Klasse selbst ist eine Unterklasse der zu betrachtenden `ViewModel` Klasse. Die `ViewModel` Klasse fordert die benötigten Daten aus dem UA Informationsmodell über das für TheClient zu implementierende SDK durch die Klasse `Browse Service` an. Der `Browse Service` und der `DataItem Service` abstrahiert hierbei die Programmierschnittstelle (API) Aufrufe des zu implementierenden UA Stacks gegenüber dem UA Server. Die vom UA zurückgelieferten Werte werden in dem generalisierten C#-Typ `DataItem` abgelegt. Durch die Generalisierung der Daten in einem Typ, kann die Entscheidungslogik für die Auswahl der passenden Designkomponente auf einen einzigen C#-Typ fokussiert werden, was eine einfachere Programmlogik zur Folge hat. Die jeweilige View Klasse initiiert die vollständige Automatik. Dabei initiiert sie zuerst die beschriebene Automatik zum Datenabruf über das ViewModel. Anschließend kann sie für

die zurückgelieferten Daten unter Anwendung der `TemplateSelector` Klasse das richtige DataTemplate und damit die richtige Visualisierung wählen. Die `TemplateSelector` Klasse ist eine Unterklasse der `DataTemplateSelector` Klasse in Xamarin.Forms und ermöglicht die Auswahl eines DataTemplates. Oben wurde die Umwandlung von Komponenten nach solchen entsprechenden DataTemplate erläutert. Die `TemplateSelector` Klasse übschreibt die `OnSelectTemplate` Methode. Diese Methode muss dabei das DataTemplate zurückgeben, dass tatsächlich den Datenknoten repräsentieren soll. In diesem Fall muss der Datenknoten also über eine Verbindung zur `TemplateHandling` Klasse geschickt werden, um ein wie bereits beschrieben passendes DataTemplate für den Datenknoten zu erhalten. Die Abbildung zeigt stattdessen die Methodenweiterleitung über das Interface `ViewModel Basis`, welches vom jedem `ViewModel` implementiert wird, zum `ViewModel`. Dies ist als Entwurfsentscheidung gewollt, genauso wie die Vererbung zwischen `TemplateHandling` Klasse und `ViewModel`. Dies ermöglicht nämlich für jede `ViewModel` Klasse und damit für jede `View` Klasse eine Anpassung der standardmäßig ausgelieferten Komponente in der jeweiligen `ViewModel` Klasse, durch Überschreibung der entsprechenden Methode aus dem `TemplateHandling`.

Die vollständige Implementierung von TheClient als Quellcode ist auf beiliegender DVD zu finden.

6. Schlussbetrachtung

Die Erkenntnisse dieser Arbeit zeigten, dass ein Abruf, Visualisieren und Modifizieren von Daten unabhängig des Geräteherstellers und der eigentlichen Information unter Berücksichtigung etablierter Geräteintegrationstechnologien nicht nur theoretisch, sondern durch die prototypische Implementierung auch praktikabel plattformübergreifend möglich ist.

Die wichtigste Erkenntnis dieser Arbeit ist der besondere Stellenwert des UA Toolkits, dass den Abruf und die Modifizierung von Daten ermöglicht und so die Grundlage für die Visualisierung stellt. Die Nutzung eines bestehenden UA Toolkits, ist aufgrund der Komplexität des UA Standards unumgänglich, um zeitnah eine Visualisierungsanwendung zu entwickeln. Gleichzeitig existiert derzeit kein UA Toolkit dass die Minimalanforderungen zur plattformübergreifenden Anwendungsentwicklung mit Xamarin erfüllt. Bestehende Teilmengen von auf dem Markt verfügbare Toolkits, sogenannte Stacks, reduzieren die Komplexität der Implementierung durch Abstraktion der im Standard definierten Anwendungsfälle, schaffen gleichzeitig aber neue Komplexität durch Wissenstransfer. So müssen die in der Praxis in der Regel durch ein Toolkit abstrahierte Anwendungsfälle des UA Standard durch Anwendungsentwickler dennoch verstanden werden, um mit einem entsprechenden selbst entwickelten SDK die Funktionalitäten des Standards vollständig von der eigentlichen Geschäftslogik der Anwendung zu abstrahieren. Auch wenn die prototypische Implementierung aus Gründen dieser Komplexität der Entwicklung eines SDKs, derzeit ausschließlich unter Xamarin lauffähig ist, zeigten die exemplarischen Modifikationen gemäß der in dieser Arbeit beschriebenen Modifikationsstrategien, dass die Plattform- unabhängigkeit mit ausreichend Ressourcen zu realisieren ist.

Neben dem Abruf und Modifizieren von Daten durch das Toolkit zeigt eine weitere primäre Erkenntnis dieser Arbeit die Notwendigkeit einer dynamischen Visualisierung der Informationen. Diese Notwendigkeit besteht aufgrund der in der Theorie unendlich möglichen Typenvielfalt, die durch den UA Standard nicht beschränkt werden. Die beschriebene theoretische Konzeptlösung und die anschließende erfolgreiche praktische Implementierung, zeigten dass die dynamische Visualisierung von Daten aus dem UA Informationsmodell keine nennenswerten Probleme verursachen und mit Xamarin plattformübergreifend Ressourcen einsparen.

Durch die standardmäßige Unterstützung zur Integration etablierter Geräteintegrationstechnologien durch den UA Standard und die Abstraktion dieser auf Konsumentenseite durch genormte Dienstschnittstellen, ergeben sich für den Anwendungsentwickler auf Konsumentenseite keine nennenswerten Probleme. Dies setzt allerdings voraus, dass die Standardisierungsgremien der zu integrierenden Technologien die Integration gemäß dem UA Standard ordnungsgemäß realisieren. Die in dieser Arbeit betrachtete etablierte FDT Technologie zeigte mit dem *OPC UA Informationsmodell für FDT* eine gelungene aber derzeit noch nicht vollständig ausgereifte Integration.

Im Kontext von Industrie 4.0 ist der Autor der Meinung, dass die UA Technologie zum semantischen Datenaustausch unternehmensweit als geeignet zu betrachten ist, besonders um die homogene technologische Landschaft in Unternehmen ohne größeren Ressourceneinsatz durch Integration bestehender Technologien zu vereinheitlichen und so die Basis für die künstliche Intelligenz und damit schließlich die vierte industrielle Revolution aus technischer Sicht national und international einzuläuten. Damit dies zeitnah geschehen kann, besteht allerdings erheblicher Bedarf den theoretischen UA Standard in einer technischen plattformübergreifenden Implementierung für als Toolkit zu realisieren. Erst wenn dies geschehen ist, können Anwendungsentwickler ihre Anwendungen realisieren, ohne erheblichen Ressourcenbedarf in das Verständnis des UA Standards.

A. Technologien Exkurs

Die nachfolgenden Unterkapitel beschreiben die in dieser Arbeit genutzten Technologien mit UA für den unabhängigen und standardisierten Datenaustausch und die entsprechende Datenmodifizierung sowie FDT für den etablierten Datenaustausch auf horizontaler Ebene und das *OPC UA Informationsmodell für FDT*, welches als Integrationsmittel für FDT nach UA dient, weiter im Detail. Die nachfolgenden Unterkapitel dienen somit als Ergänzung und Nachschlagewerk zu den in dieser Arbeit erwähnten Teilaspekten der Technologien. Hierbei sind die Unterkapitel lediglich als Abriss der komplexen Beschreibungen zu verstehen. Die detaillierteste Beschreibung der Unterkapitel findet sich in den Original Standards der Technologien.

A.1. Open Platform Communications Unified Architecture

UA ist ein Standard zum plattformunabhängigen und herstellerübergreifenden Informationsaustausch. [13, 31] Durch das Bereitstellen von Protokollen und Diensten regelt die Technologie den Austausch von Informationen zwischen Anwendungen. Durch die Technologiedienste können reichhaltige Informationsmodelle publiziert werden, um komplexe Daten zwischen unabhängig entwickelten Anwendungen mit den definierten Protokollen der Technologie auszutauschen. [13] Es wird somit explizit zwischen Mechanismen für den Informationsaustausch und den auszutauschenden Daten als solche unterschieden. [13, 31] Im Rahmen von UA werden Informationen in die folgenden Kategorien gegliedert: Prozessdaten, Alarm- und Ereignissignale, historische Daten und Kommandos. [13] Gerätehersteller können Gerätedaten derer Geräte auf Grundlage des im UA Standard verankerten Standardinformationsmodell abbilden. [30] Der architektonische Aufbau einer UA Anwendung ist in

Abbildung A.1 illustriert. Entsprechend diesem Aufbau werden Daten nach dem Client-Server Konzept zwischen Anwendungen ausgetauscht. Eine UA Anwendung kann hierbei ein UA Server oder ein UA Client sein. Ein UA Server ist eine Anwendung, die Information für andere Anwendungen bereitstellt. In Analogie hierzu ist ein UA Client eine Anwendung die UA Informationen konsumieren möchte. [31] Mit der lassen sich Gerätedaten neben der Abbildung dieser im Standardinformationsmodell auch in erweiterten Informationsmodellen definieren, wie Abbildung A.1 zeigt. [30] Eine Vielzahl an Organisationen haben bereits damit begonnen, solche erweiterten Informationsmodelle für deren Standards zu spezifizieren. Einige dieser Organisationen haben diesen Prozess bereits beendet und bieten Geräteherstellern, die Möglichkeit zur Nutzung dieser Modelle zur Integration der Geräte in UA. [13]

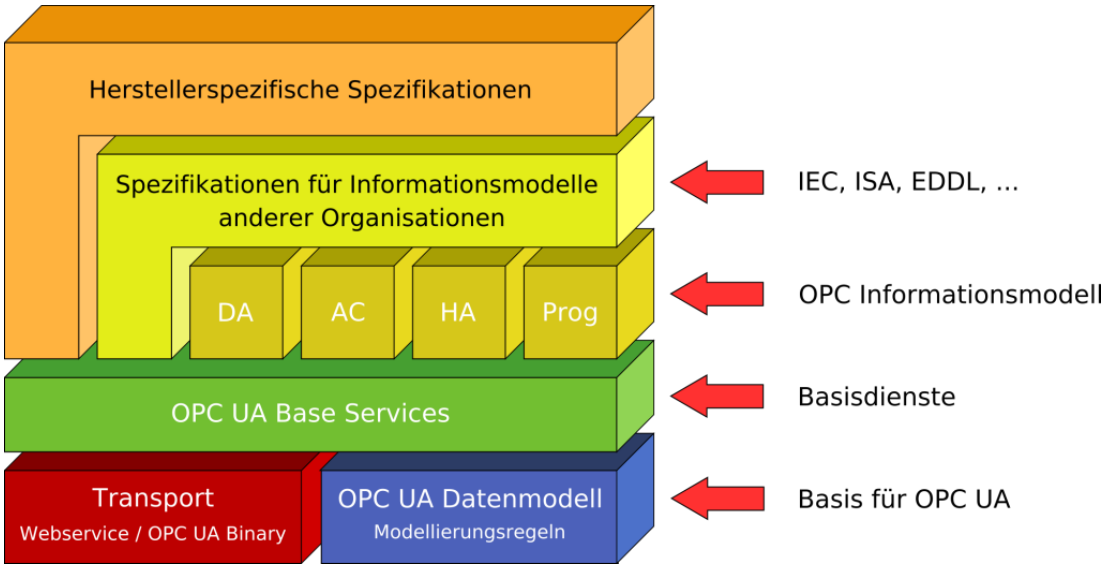

Abbildung A.1.: Architektur der Open Platform Communications Unified Architecture Technologie [89]

Durch die hohe Flexibilität ist UA die ideale Grundlage zur Kommunikation in einer voll vernetzen, datengetriebene Industrie 4.0 Fabrik. Die Technologie ermöglicht die vollständige vertikale Integration aller informationsverarbeitenden Systeme durch die vollständige Abbildung dieser ohne ständige notwendige Protokollumsetzungen. [13, 90] Als solche ist die Technologie als De-facto-Standard im Kontext der vierten industriellen Revolution im Referen- zarchitekturmodell Industrie 4.0 gelistet. [25] Obwohl es sich um eine Branchen

neutrale Technologie handelt, findet UA derzeit die meiste Verbreitung in der Automatisierungstechnik. In dieser wird sich UA langfristig auch durchsetzen. [13] Mit der Unabhängigkeit von Kommunikationstechnologie der Hersteller, Branchen, Betriebssysteme und Programmiersprachen werden alle Anforderungen für Industrie 4.0 gedeckt. [13] Alle über die UA Technologie verbundenen Geräte und Anwendungen kommunizieren über einen festen Satz von 37 Dienstschnittstellen miteinander. [13, 24, 25] Die hohe Skalierbarkeit der Technologie ermöglicht es, dass UA sowohl auf eingebetteten Geräten direkt auf einem Sensor als auch zum Ausbau von unternehmensweiten Datenverwaltungsservern auf Mainframe-Rechnern genutzt wird. [13, 26] Beispiele für einen praktikablen Einsatz zeigt das *Fraunhofer-Anwendungszentrum Industrial Automation (IOSB-INA)* gemeinsam mit dem *Institut für Industrielle Informationstechnik* der Hochschule Ostwestfalen-Lippe. In einer Unternehmung implantierten diese einen UA Server direkt auf einem Chip. Ein weiterer Anwendungsfall des Unternehmens AREVA zeigt wie ein UA Server in einem Sensor von Überwachungsgeräten implantiert wird. Diese Lösung findet heute zur Überwachung hoch kritischer Systeme in der Nuklearbranche Anwendung. Zusammenfassend ermöglicht UA eine sichere, horizontale und vertikale Kommunikation vom Sensor bis hin zum IT-System auf höheren Ebenen. [13]

A.1.1. Informationsmodellierung

Wie Kapitel 2 detailliert beschreibt, bildet die Kommunikation zwischen den einzelnen am Wertschöpfungsprozess beteiligten Objekten die Grundlage für ein Netz intelligenter Produkte und Produktionskomponenten. [11] Um diese Objekte zu modellieren bedarf es einer beliebig komplexen Abbildung der möglichen Informationsinhalte eines solchen Objekts. Der UA Standard beschreibt hierfür ein voll vermaschtes Netz an Datenknoten, wie Abbildung A.2 abstrakt zeigt. [40]

Die Datenknoten sind als Objekte organisiert um ein objektorientiertes Programmierparadigma zu ermöglichen. Neben den Daten enthält ein Datenknoten zusätzlich Metainformationen, über die im Datenknoten gespeicherten Daten. [31, 40] Der Aufbau beliebiger Objektstrukturen innerhalb dieses voll ver-

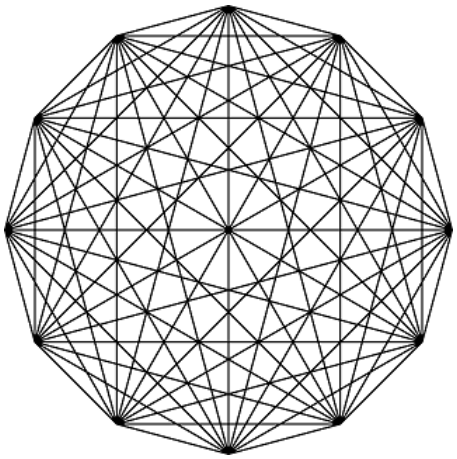

Abbildung A.2.: Voll vermaschtes Netzwerk an Datenknoten [91]

maschten Netzwerks an Datenknoten wird durch die Verwendung von Referenzen zwischen einzelnen Instanzen der Datenknoten ermöglicht. Mit Hilfe dieser Referenzen wird es einer Client Anwendung ermöglicht, sich durch ein Netz vom UA Server bereitgestellten Datenknoten zu navigieren, welche als Objektstruktur bereitgestellt ist. [13] Ein Datenknoten im Informationsmodell wird durch Attribute beschrieben. Diese Attribute unterscheiden sich jeweils nach Verwendungszweck des Datenknotens. Ein Datenknoten ist von einem bestimmten Typ. Ein Typ wird in UA durch die sogenannte Datenknoten Klasse beschrieben. Die wichtigsten Datenknotentypen sind Objekte, Variablen und Methoden. In Analogie zur objektorientierten Programmierung können Objekte Variablen und Methoden beinhalten. Methoden können Ereignisse auslösen. Eine Variable repräsentiert einen Wert. Eine Methode liefert ein Resultat zurück, nachdem diese vom Client aufgerufen wird. [40]

Neben der Abbildung von Informationen in das von UA definierte Standardinformationsmodell stellt die Migration und Integration bestehender Informationsmodelle eine weitere Hürde für den Erfolg von Industrie 4.0 dar. So soll es eine Möglichkeit geben, die Informationsmodelle etablierter Technologien in das im Wertschöpfungsprozess allgemeingültige Informationsnetzwerk zu integrieren, welches im geeignetsten Falle auf UA basiert. [13] Die Möglichkeit zur Erweiterung des von UA festgelegten Standardinformationsmodells ist ebenfalls im UA Standard niedergeschrieben. [13, 31]

Dies macht UA besonders interessant für andere Standardisierungsgremien. Besonders für solche, die sich in der Vergangenheit darauf konzentrierten, die Kommunikation auf horizontaler Ebene zu standardisieren. In diesem Rahmen arbeitet die OPC Foundation bereits mit einer Vielzahl anderer Standardisierungsorganisationen zusammen, die in Abbildung A.3 aufgelistet sind. Hier sei *OPC UA for Analyser Devices*, *Field Device Integration*, *ISA95*, *MTConnect*, *BACnet*, *PLCOpen* und *OPC UA for Devices* [41] erwähnt. [13]

Abbildung A.3.: Organisationen verschiedener Branchen, die ihre Technologien in Open Platform Communications Unified Architecture integrieren [92]

Es können so etablierte Standards auf horizontaler Ebene der Automatisierungspyramide weiterhin genutzt werden, ohne die vollständige Infrastruktur im Kontext der vierten industriellen Revolution zu adaptieren. Dies bedeutet, dass sowohl Software- als auch Hardwarekomponenten auf horizontaler Ebene innerhalb der Produktionsdomäne nicht ausgetauscht werden müssen. Dieses Verfahren ist im Gegensatz zur vollständigen Substitution der Produktionsinfrastruktur als geeigneter zu betrachten. So können neben massiven Eingriff in den Produktionsablauf und die damit verbundenen Stillstandskosten der einzelnen Produktionsstellen auch in der Vergangenheit getätigte Investitionen gesichert werden. [15, 25] Ein erweitertes Informationsmodell beantwortet so beispielhaft die Fragestellung 'Wie gibt sich ein Temperatursensor zu erkennen?' oder 'Wie gibt sich eine Ventilsteuerung zu erkennen?'. Erweitert werden kann das allgemeingültige UA Informationsmodell durch Objekte, Methoden, Variablen und Ereignisse oder durch die Definition, welche Objekte, Methoden, Variablen und Ereignisse im allgemeingültigen UA Informationsmodell für Konfiguration, Initialisierung, Diagnose und Laufzeit zuständig sind. Zusätzlich

wird die Semantik der Daten beschrieben, um Informationen darüber zu erhalten, welche Daten ausgetauscht werden.

Die Frage nach wie soll der Datenaustausch vonstattengehen, ist somit nicht länger die Aufgabe der Technologien auf horizontaler Ebene, sondern Bestandteil von UA. [13]

A.1.2. Transport und Sicherheit

Die serviceorientierte Architektur auf hoher Abstraktionsebene und die Nutzung etablierter und standardisierter Transportstandards auf unterer Abstraktionsebene in UA befriedigen die in Industrie 4.0 definierten Anforderungen für Transport und Sicherheit. Der Transport kann derzeit laut UA Standard über TCP oder alternativ über HTTP und Simple Object Access Protocol (SOAP) erfolgen. Der Standard definiert allerdings eine Unabhängigkeit zur Transportschicht, sodass weitere Protokollbindungen folgen können. Derzeit ist beispielsweise die zusätzliche Kommunikation mittels Publish/Subscriber Architektur in Planung. Verschiedene im Standard definierte Sicherheitsmechanismen sorgen für einen stabilen Transport, der bei Unterbrechungen innerhalb der Transportschicht nicht zum Ausfall der zu übertragenden Daten führt. Beispielhafte Mechanismen sind die Verwendung von Puffer-Technologien oder die Anpassung der Übertragung je nachdem, ob es sich um einen kabelgebundenen oder mobilen Client handelt. [13]

Eine der wesentlichsten Anforderungen in Industrie 4.0 ist die Sicherheit der Datenübertragung und die damit in Verbindung stehende Authentifizierung von Anwendern. Um diesen Anforderungen nachzukommen, definiert UA die Verwendung von x509-Zertifikaten zur signierten Übertragung. Zur Authentifizierung auf Anwendungsebene kann das Netzwerkprotokoll Kerberos oder alternativ die Verwendung einer Benutzername/Passwort-Kombination verwendet werden. Somit gewährleistet die UA Technologie die drei Aspekte: Authentifizierung, Signierung von Nachrichten und Verschlüsselung mittels Secure Sockets Layer Mechanismen. [13]

A.1.3. Discovery

Erst mit einer schnellen Anpassung an neue Marktbedingungen und der Möglichkeit verschiedene Produktvarianten umzusetzen, können selbst organisierende Produktionsprozesse gewährleistet werden. Dies macht unter Umständen einen Austausch oder die Orchestrierung von Produktionskomponenten wie Maschinen zur Laufzeit des Produktionsprozesses notwendig. [13, 20] Für diese weitere Anforderung zur Realisierung von Industrie 4.0 definiert der UA Standard verschiedene, sogenannte Discovery Konzepte. Diese Konzepte dienen zur Bekanntmachung von UA Teilnehmern. [38] Dies ist besonders dann von großer Bedeutung, wenn sich eine Vielzahl von UA Teilnehmer in der administrativen Domäne aufhalten. [1] Durch Einbettung von UA Servern direkt innerhalb von Feldgeräten steigt die Anzahl von UA Anwendungen noch deutlich. [32, 93] Der detaillierte Prozess des Discovery wird im Standard erläutert. [38] Dabei adressieren die Konzepte zwei Problembereiche, die durch den beschriebenen Discovery-Prozess gelöst werden. Die erste Problematik ist das Auffinden eines UA Servers durch einen UA Client innerhalb des Netzwerkes. Die zweite Schwierigkeit skizziert wie Verbindungsinformationen, darunter auch Sicherheitsinformationen, von einem aufgefundenen Server im Netzwerk konsumiert werden können, um eine Verbindung mit diesem herzustellen. [38, 94] Einige der Lösungskonzepte verwenden Discovery Server.

Ein Discovery Server ist eine Anwendung, welche eine Liste von innerhalb des Netzwerkes verfügbare UA Servern verwaltet. Es wird zwischen globalen und lokalen Discovery Servern unterschieden. Ein globaler Discovery Server ist innerhalb der vollständigen administrativen Domäne erreichbar. Der lokale Discovery Server hingegen befindet sich in der Regel auf derselben Workstation wie auch die UA Anwendungen. Der größte Wirkungsbereich eines lokalen Discovery Server ist innerhalb desselben Subnetzwerkes. Wenn von Endpunkten die Sprache ist, unterscheidet man im Kontext des Discovery Prozess zwischen zweierlei Typen von Endpunkten. Dem Discovery-Endpunkt und dem Sitzungsendpunkt. Der Discovery Endpunkt ist die URL, welche die Verbindungsinformationen eines UA Servers zurückliefert. [38] Dem entgegen steht der Sitzungsendpunkt. Der Sitzungsendpunkt ist eine URL, mit der eine Ver-

bindung zum Server aufgebaut und eine Sitzung eröffnet werden kann. Diese URL kann aus einer Menge verfügbarer Sitzungsendpunkte extrahiert werden, welche der UA Server anbietet. Die Sitzungsendpunkte können mit Erhalt der Verbindungsinformationen abgegriffen werden. [1, 36, 37, 38]

Die UA-Serveranwendung benötigt zur Laufzeit einen Wirt. Dies ist in der Regel eine in der administrativen Domäne eingebettete Workstation. Diese bietet mit der Laufzeitumgebung eine Systemumgebung für die UA-Serveranwendung. Dabei ist die Systemumgebung nicht auf die Ausführung einer einzelnen UA-Anwendung beschränkt. So sind für den Betrieb von UA Anwendung auf einer Workstation die verschiedensten Varianten möglich: Einzelner oder mehrfacher UA Client, einzelner oder mehrfacher UA Server oder sowohl ein als auch mehrere UA-Clients und UA-Server gleichzeitig. Die Anzahl der Workstations, die eine UA-Anwendung ausführen, können sich im gleichen Subnetzwerk oder aber an ganz verteilten Standorten innerhalb der administrativen Domäne befinden. [38] Durch die Flexibilität der unabhängigen Standortwahl innerhalb der administrativen Domäne ergeben sich die verschiedenen skizzierten Problembereiche. Für diese sind im Standard die erwähnten Lösungskonzepte erläutert, die auf der Kommunikation mittels TCP basieren:

- Einfacher Discovery: Der Discovery-Endpunkt des UA Servers auf der Workstation ist bekannt. Die vom UA Server bereitgestellten Sitzungsendpunkte können erfragt werden. Eine entsprechende Sitzung kann durch eine der abgefragten Sitzungsendpunkte erzeugt werden. [1]

- Normaler Discovery: Die IP-Adresse oder der Hostname der Workstation, die für den UA Server als Wirt dient, ist bekannt. Der Discovery EndpunktEndpunkt des UA Servers auf der Workstation aber nicht. Der Local Discovery Server (LDS), der sich auf der Workstation befindet, wird kontaktiert. Bei Existenz des LDS, liefert dieser den Discovery-Endpunkt des UA Servers zurück. Anschließend kann wie beim einfachen Discoveryverfahren werden, um so eine Sitzung mit dem Server zu erzeugen. [1, 38]

- Hierarchischer Discovery: Es ist weder der Discovery-Endpunkt des UA Servers bekannt, noch die IP-Adresse oder Hostname der Workstation,

welche die Anwendung beherbergt. Somit wird der Global Discovery Server (GDS) befragt, der sich innerhalb der administrativen Domäne befindet. Dieser liefert eine Liste mit allen bei dessen registrierten LDS und UA Server. Befindet sich der gewünschte UA Server in dieser ersten Hierarchiestufe der vom GDS zurückgelieferten Liste, kann das Verfahren wie beim einfachen Discovery fortgeführt werden. Wird ein LDS zur weiteren Suche nach dem passenden UA Server genutzt, kann das Verfahren wie beim normalen Discovery fortgeführt werden. [1]

A.1.4. Serversitzung

Eine Serversitzung definiert eine logische und dauerhafte Verbindung zwischen einem UA Server und einem UA Client. Eine solche Serversitzung dient als kontextuelles Modell über Dienstaufrufe hinweg für Zustände, welche in diesem Modell abgebildet werden. Beispiele für zustandsbehaftete Informationen sind Nutzeranmeldeinformationen, Datenabonnements oder Informationen zur Fortsetzung des Dienstes für Operationen, welche mehr als eine Anfrage umfassen. [94] Hierbei ist eine Serversitzung völlig unabhängig von dem darunterliegenden Kommunikationsprotokoll.

Während die im UA Stack implementierten Transportmechanismen die Anforderungen an Sicherheit, Flexibilität und Zuverlässigkeit der Datenkommunikation gewährleisten, wird die Serversitzung auf Anwendungsebene dafür verwendet, um den Anwender zu authentifizieren. [36, 94]

Ein Fehler innerhalb der Transportschicht zieht somit nicht zwingend die Beendigung der Serversitzung nach sich. [94] Durch die Erzeugung verschiedener Serversitzungen können UA Clients verschiedene Sichten und Berechtigungen auf das Informationsmodell erhalten. Beispielsweise könnte eine speicherprogrammierbare Steuerung eine Million Variablen verwalten, aufgrund der Sicht können aber nur 500 Variablen visualisiert werden. In einem anderen Anwendungsfall könnte ein MES fünf Variablen halten, welche allesamt aufgrund eingeschränkter Berechtigung im Kontext der Sitzung nicht verändert werden dürfen. [13]

A.2. Field Device Tool

In den heutigen Produktionsumgebungen wird in der Regel Feldgerät vieler verschiedener Hersteller integriert. Dies bedeutet gleichzeitig steigender Aufwand für Wartungsaufgaben wie Installation, Versionsverwaltung oder die einfache Bedienung dieser Geräte. Um dem stetigen Aufwandsanstieg gerecht zu werden, werden Technologien verwendet, welche eine standardisierte Geräteintegration in die Systemlandschaft ermöglichen. Eine dieser Geräteintegrationstechnologien ist FDT. [43] Allgemein gesprochen ermöglicht FDT den Datenaustausch zwischen Feldgeräten und Automatisierungssystemen. [41]

Der FDT Standard beschreibt detailliert die Kommunikations- und Konfigurationsschnittstellen von Feldgeräten und Hostsystemen. [41, 44] Die Technologie kann wie auf Abbildung A.4 zu sehen, durch drei Hauptkonzepte beschrieben werden: DTM, eine Rahmenanwendung und Schnittstellen. Das von FDT erklärte Ziel hierbei ist die Bereitstellung einer gemeinsamen grafischen Benutzeroberfläche, über die jedes Feldgerät konfiguriert, bedient und gewartet werden kann, unabhängig des Herstellers, Gerätetyps oder Kommunikationsprotokoll des Geräts. [44]

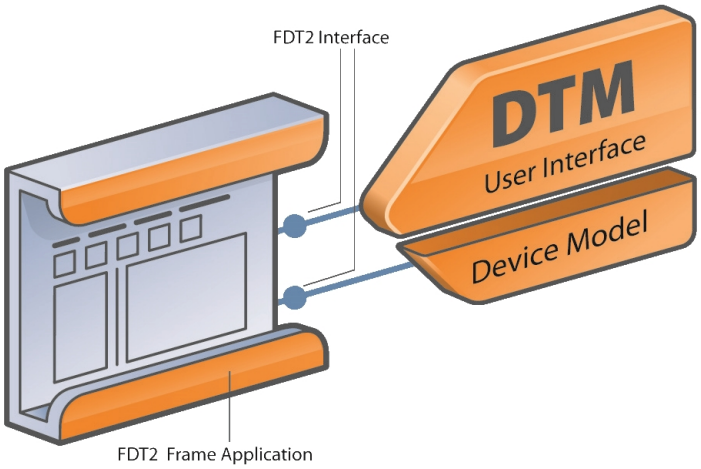

Abbildung A.4.: Die drei Hauptkonzepte von Field Device Tool: Device Type Manager, Rahmenanwendung, Schnittstellen [44]

Ein DTM ist eine Softwarekomponente, die ein oder mehrere physische Feldgeräte repräsentiert. [41] Er wird in aller Regel vom Hersteller des physischen

Feldgeräts entwickelt. Hierzu wird die Struktur des physischen Feldgeräts auf die standardisierte Struktur der FDT Komponente des DTMs abgebildet. Damit wird es für einen Datenkonsumenten ermöglicht auf Geräteparameter, Gerätekonfiguration und Diagnoseinformationen über den DTM auf dem physischen Feldgerät zuzugreifen. Es können zudem Steuerungsbefehle über den DTM an das physische Gerät gesendet werden. Ein DTM kann in verschiedenen Variationen auftreten. Die verschiedenen Varianten können zwei Gruppierungen zugeordnet werden: Geräte-DTM oder Kommunikations-DTM. Neben dem bereits beschriebenen Geräte-DTM bildet der Kommunikations-DTM nicht die Geräte, sondern die umliegende Topologie ab. Dies kann beispielsweise die Netzwerkkarte eines Systems sein. [44] Bei dem Kernkonzept der Rahmenanwendung handelt es sich um eine Softwareumgebung. In dieser Softwareumgebung können die verschiedenen DTMs integriert und visuell abgebildet werden. [41, 46] Die Rahmenanwendung kann als Laufzeitumgebung der DTMs betrachtet werden. [45, 46] Die Laufzeitumgebung und damit die Rahmenanwendung dient als Mittel zur Verwaltung und Koordination der DTMs. [35, 43] Die Interaktion zwischen der Rahmenanwendung und einem DTM findet dabei über die in FDT standardisierten Schnittstellen statt. [43, 44, 47]

Alle Anwendungen welche diese in FDT spezifizierten Schnittstellen unterstützen können in der Theorie als Rahmenanwendung bezeichnet werden. [47] Je nach geplantem Einsatz der Anwendung darf sich diese unter Umständen in Verhalten und Funktion unterscheiden. Durch die Integration der Schnittstellen und dem zum physischen Gerät dazugehörigen DTM, kann eine Rahmenanwendung dennoch jede Art von Gerätetyp ohne Wissen über das physische Gerät oder gar des Feldbusses verwalten. [46]

A.3. OPC UA Informationsmodell für FDT

In Kapitel A.1.1 wurde erläutert, dass Informationen, die von etablierten Technologien auf horizontaler Ebene bereitgestellt werden, sich in das UA Standardinformationsmodell integrierten lassen. [41] Als eine der ersten Standardisierungsorganisationen bildete die FDT Group gemeinsam mit der OPC Foun-

dation eine Arbeitsgruppe zur Spezifikation einer solchen Informationsmodellerweiterung. Das Ermöglichen einer einheitlichen Sicht auf die Geräte, ohne die Abhängigkeit einer unterliegenden Geräteprotokolltechnologie ist dabei das erklärte Ziel. [41, 42]

Resultat ist die Spezifikation *OPC UA Informationsmodell für FDT*, welche auf der Basisspezifikation von UA aufbaut und FDT an UA anbindet. [41, 42] In diesem Zusammenhang übernimmt FDT weiterhin die Aufgabe der Geräteverwaltung für die unterschiedlichsten Feldbusse und Netzwerke sowie die für das Gerät durch virtuelle Geräterepräsentanten (DTMs). So schafft die Spezifikation eine Brücke über UA zu Systemen, die Daten von den Feldgeräten benötigen. Gleichzeitig substituiert die Spezifikation im Kontext dieser Integration das FDT Informationsmodell. [41, 42]

Hauptanwendungsfälle des gemeinsam spezifizierten Modells sind die Gerätekonfiguration und Diagnose. Das Informationsmodell ermöglich dennoch auf alle durch den entsprechenden DTM abgebildeten Informationen des repräsentierten Geräts zuzugreifen. [41]

Zur Abbildung der FDT Informationen in das Standardinformationsmodell von UA wurden von der Arbeitsgruppe die entsprechenden Objekte nach Identifikation sorgfältig ausgewählt. An einigen Stellen wurde hierzu das UA Gerätemodell, welches als Vorlage für die Spezifikation dient, erweitert in Form von zusätzlichen Objekttypen und Prozeduren. Damit werden zusätzliche als wichtig bestimmte Informationen von FDT berücksichtigt. [41, 42] In der Spezifikation [41] zur Erweiterung des Informationsmodells, wird an einigen Stellen FDT in der Version 2 vorausgesetzt. Zusätzlich basiert das FDT Testszenario auf FDT2. Wenn in der Arbeit von FDT geschrieben wird, so wird damit die Version zwei dieser Technologie angezielt.

B. Anwendungsfälle der Technologien

Die nachfolgenden Unterkapitel zeigen die Anwendungsfälle der in dieser Arbeit genutzten Technologien als Gesamtauflistung und im Detail.

B.1. Open Platform Communications Unified Architecture Anwendungsfälle

In diesem Unterkapitel werden die aus dem Standard extrahierten Anwendungsfälle der UA Technologie aufgelistet und beschrieben.

Tabelle B.1.: Generische und der Detaillierungsanalyse entsprechende extrahierte Anwendungsfälle der Open Platform Communications Unified Architecture Technologie (basiert auf Informationen aus [36])

Anwendungsfall	**Beinhalteter / Spezifischer Anwendungsfall**	**Beinhalteter / Spezifischer Anwendungsfall**	**Beschreibung**
Server finden			Ermöglicht verfügbare UA Server in einer administrativen Domäne zu finden und Sitzungsendpunkte zum Aufbau einer Sitzung der gefunden UA Server abzugreifen
	Local Discovery Server (LDS)		Ermöglicht verfügbare UA Server durch einen Local Discovery Server (LDS) zu finden
		LDS auf Remote Host	Ermöglicht Verbindungsinformationen zu verfügbaren UA Servern, von einem entfernten Local Discovery Server zu erhalten

Fortsetzung auf nachfolgender Seite

Tabelle B.1 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter / Spezifischer Anwendungsfall**	**Beinhalteter / Spezifischer Anwendungsfall**	**Beschreibung**
		LDS auf Client Host	Ermöglicht verfügbare UA Server von einem Local Discovery Server zu erhalten, der auf der gleichen physikalischen Maschine wie die UA Client Anwendung ausgeführt wird
		Multicast Domain Name System (mDNS)	Ermöglicht verfügbare UA Server durch das Multicast Domain Name System zu finden
	Eingabe der Server URL		Ermöglicht den gewünschten UA Server durch Eingabe der Discovery URL des gesuchten UA Servers durch den Anwender zu finden
	Global Discovery Server (GDS)		Ermöglicht verfügbare UA Server durch einen Global Discovery Server (GDS) zu finden

Fortsetzung auf nachfolgender Seite

Tabelle B.1 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter / Spezifischer Anwendungsfall**	**Beinhalteter / Spezifischer Anwendungsfall**	**Beschreibung**
	Sitzungsendpunkte erfragen		Ermöglicht die Sitzungsendpunkte eines UA Servers zu erfragen, die zum Aufbau einer Sitzung benötigt werden
Verbindungsverwaltung			Gewährleistet eine sichere, flexible und zuverlässige Datenkommunikation zwischen UA Anwendungen
	Netzwerktransport		Ermöglicht Nachrichten zwischen UA Anwendung auszutauschen
	Sicherheitsabwicklung		Ermöglicht die kryptografische Absicherung von Nachrichten
	Sitzungsabwicklung		Ermöglicht Benutzern einer UA Clientanwendung die Authentisierung
Informationen im Adressraum finden			Ermöglicht das Finden von verschiedenen Typen von Daten im Adressraum eines UA Servers

Fortsetzung auf nachfolgender Seite

Tabelle B.1 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter / Spezifischer Anwendungsfall**	**Beinhalteter / Spezifischer Anwendungsfall**	**Beschreibung**
	Navigieren durch Datenpunkte		Ermöglicht die Navigation durch die verschiedenen hierarchisch geordneten Datenpunkte des UA Informationsmodells
	Metainformationen von Datenpunkten lesen		Ermöglicht das Lesen von Metainformationen der Datenpunkte im UA Informationsmodell
Lesen und Schreiben von Daten und Metadaten			Ermöglicht das nicht zyklische Lesen und Aktualisieren von Datenknoten aus beziehungsweis in einem UA Informationsmodell, welches von einer UA Serveranwendung angeboten wird

Fortsetzung auf nachfolgender Seite

Tabelle B.1 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter / Spezifischer Anwendungsfall**	**Beinhalteter / Spezifischer Anwendungsfall**	**Beschreibung**
	Daten lesen		Ermöglicht das nicht zyklische Lesen von Daten der Datenknoten aus einem UA Informationsmodell, welches von einer UA Serveranwendung angeboten wird
	Metadaten lesen		Ermöglicht das nicht zyklische Lesen von Metadaten der Datenknoten aus einem UA Informationsmodell, welches von einer UA Serveranwendung angeboten wird
	Daten schreiben		Ermöglicht das Aktualisieren von Daten der Datenknoten in einem UA Informationsmodell, welches von einer UA Serveranwendung angeboten wird

Fortsetzung auf nachfolgender Seite

Tabelle B.1 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter / Spezifischer Anwendungsfall**	**Beinhalteter / Spezifischer Anwendungsfall**	**Beschreibung**
	Metadaten schreiben		Ermöglicht das Aktualisieren von Metadaten der Datenknoten in einem UA Informationsmodell, welches von einer UA Serveranwendung angeboten wird
Datenänderungen und Events abonnieren			Ermöglicht zyklisch über Änderungen von Daten oder Events informiert zu werden
	Datenänderungen abonnieren		Ermöglicht zyklisch über Änderungen von Daten informiert zu werden
	Events abonnieren		Ermöglicht zyklisch über Änderungen von Events informiert zu werden

Fortsetzung auf nachfolgender Seite

Tabelle B.1 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter / Spezifischer Anwendungsfall**	**Beinhalteter / Spezifischer Anwendungsfall**	**Beschreibung**
Server definierte Methoden aufrufen			Ermöglicht die vom durch den UA Server definierten Methoden in einem Adressraum aufzurufen
	Methode aufrufen		Ermöglicht eine definierte Methode im Adressraum eines UA Servers aufzurufen
	Liste von Methoden aufrufen		Ermöglicht alle definierten Methoden im Adressraum eines UA Servers zu erhalten
Verlauf von Daten und Events			Ermöglicht die Verwaltung von Verlauf der Daten und Events
Informationen im komplexen Adressraum finden			Ermöglicht einen an Datenpunkten reichhaltigen und dynamischen Adressraum zu durchsuchen

Fortsetzung auf nachfolgender Seite

Tabelle B.1 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter / Spezifischer Anwendungsfall**	**Beinhalteter / Spezifischer Anwendungsfall**	**Beschreibung**
Strukturmodifizierung des Serveradressraums			Ermöglicht die Struktur des Serveradressraums zu verändern
	Datenknoten erzeugen		Ermöglicht die Erzeugung eines neuen Datenknotens im Serveradressraum
	Datenknoten entfernen		Ermöglicht die Entfernung eines neuen Datenknotens im Serveradressraum
	Referenzen zwischen Datenknoten erzeugen		Ermöglicht die Verknüpfung von Datenknoten
	Referenzen zwischen Datenknoten entfernen		Ermöglicht die Entfernung von Datenknotenverknüpfungen

B.2. Field Device Tool 2 Anwendungsfälle

Der FDT Standard ordnet allen identifizierten Anwendungsfällen einer bestimmten Benutzerrolle zu. Eine Benutzerrolle gruppiert diverse Eigenschaften, welche einen realen Anwender repräsentieren soll. Der Standard illustriert die verschiedenen definierten Benutzerrollen als Akteure, die in Abbildung B.1 dargestellt werden. Wie die Abbildung B.1 zeigt, kennt FDT die drei Akteure *Beobachter*, *Experte* und *Planungsingenieur*, die auf einer Vererbungshierarchie basieren. Dargestellt ist diese Hierarchie durch die Pfeile zwischen den einzelnen Akteuren.

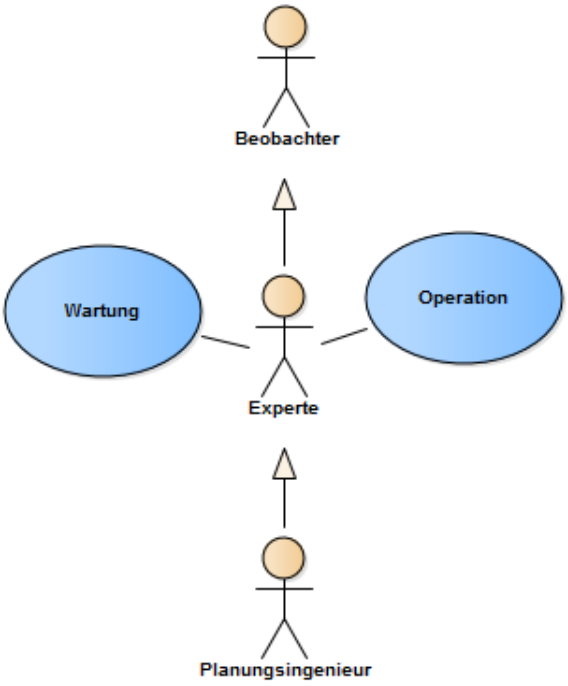

Abbildung B.1.: Definierte Akteure im Field Device Tool 2 Standard (basiert auf Informationen aus [46])

Der Akteur *Beobachter* repräsentiert einen Anwender, welcher den aktuellen Prozess in FDT beobachten kann. Abbildung B.2 zeigt diesem Akteur zugeordnete Anwendungsfälle.

Der Akteur *Experte* erbt durch die bestehende Vererbungshierarchie alle Funk-

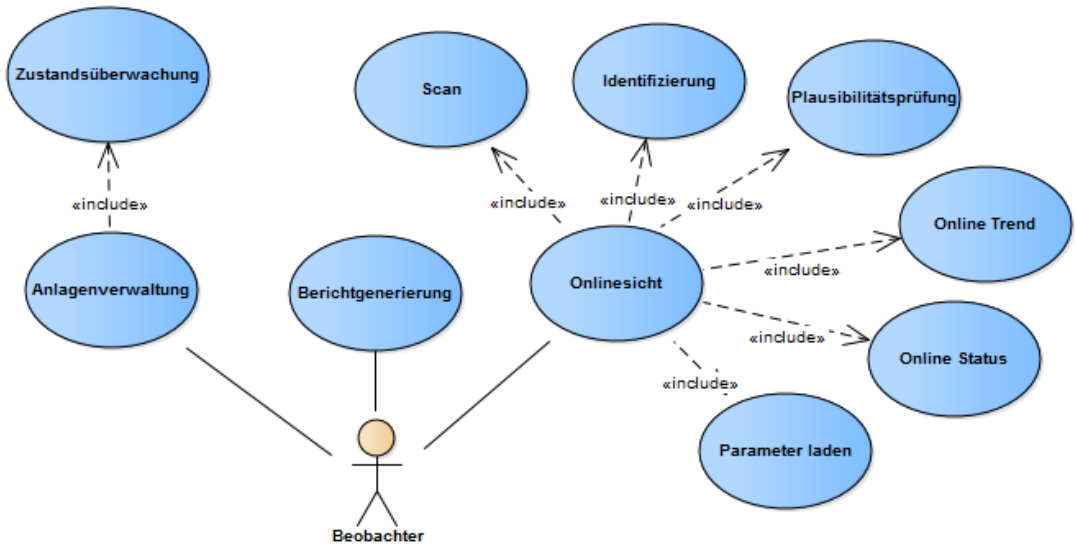

Abbildung B.2.: Akteur Beobachter und dessen zugeordnete Anwendungsfälle (basiert auf Informationen aus [46])

tionalitäten des Akteurs *Beobachter*. Neben den vererbten Anwendungsfällen, mit dem der Akteur den Prozess beobachten kann, ermöglichen die dem Akteur direkt zugeordneten Anwendungsfälle, Prozesse zu verwalten und Wartungen am FDT System vorzunehmen. Aufgrund der zwei dem Akteur direkt zugeordneten Aufgabenbereiche der *Verwaltung* und *Wartung* werden die Anwendungsfälle des Akteurs zur besseren Übersicht in die Systemsichten *Operation* und *Wartung* gegliedert.

Abbildung B.3 zeigt die der Systemsicht *Operation* direkt zugeordneten Anwendungsfälle und Abbildung B.4 die der Systemsicht *Wartung* direkt zugeordneten Anwendungsfälle.

Durch die Vererbungshierarchie überliefert der Akteur *Experte* alle Funktionalitäten an den *Planungsingenieur*. Die dem *Planungsingenieur* direkt zugeordneten Anwendungsfälle erlauben dem Akteur die Durchführung aller notwendigen Wartungsoperationen am FDT System, dem Herunterladen verifizierter Parameterdatensätze sowie die online und offline Modifikation von Parameterdatensätzen. Darüber hinaus erhält der Akteur die Befugnis zur Durchführung diverser Verwaltungsoperationen am FDT System. Abbildung B.5 zeit die dem *Planungsingenieur* direkt zuordenbare Anwendungsfälle.

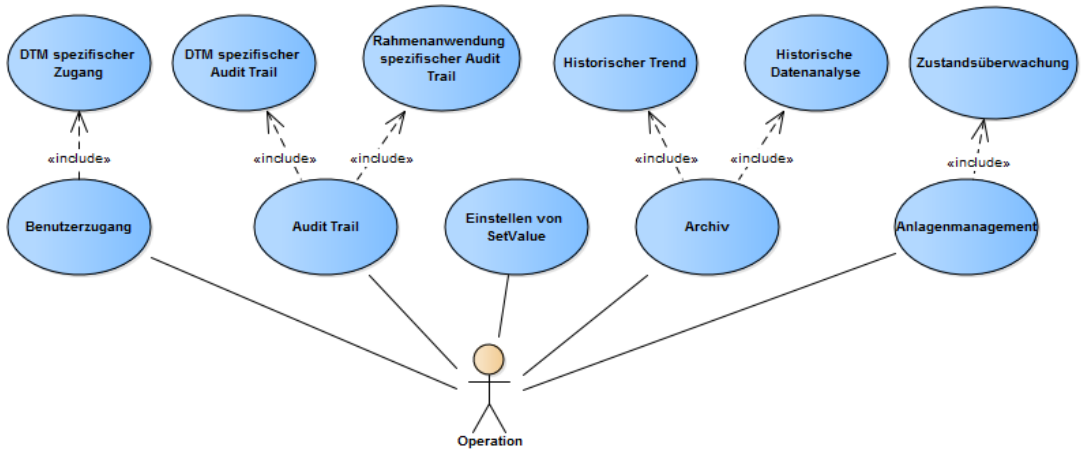

Abbildung B.3.: Systemsicht Operation und dessen zugeordnete Anwendungsfälle (basiert auf Informationen aus [46])

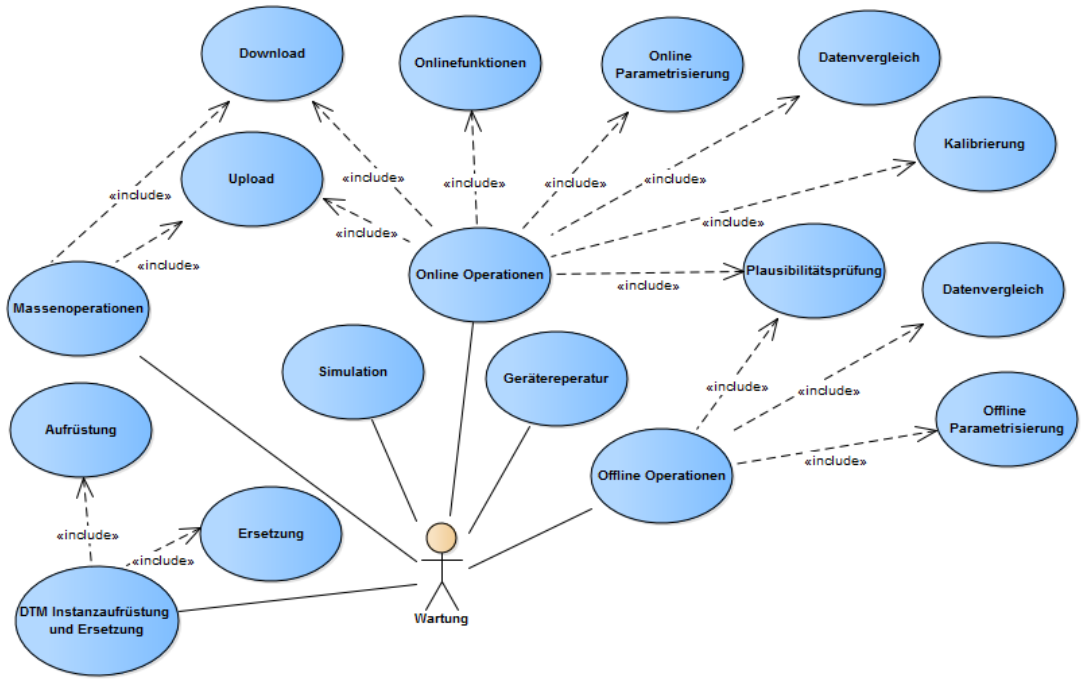

Abbildung B.4.: Systemsicht Wartung und dessen zugeordnete Anwendungsfälle (basiert auf Informationen aus [46])

Die nachfolgende Tabelle B.2 enthält alle Anwendungsfälle aller Akteure sowie eine entsprechende Detailbeschreibung dieser.

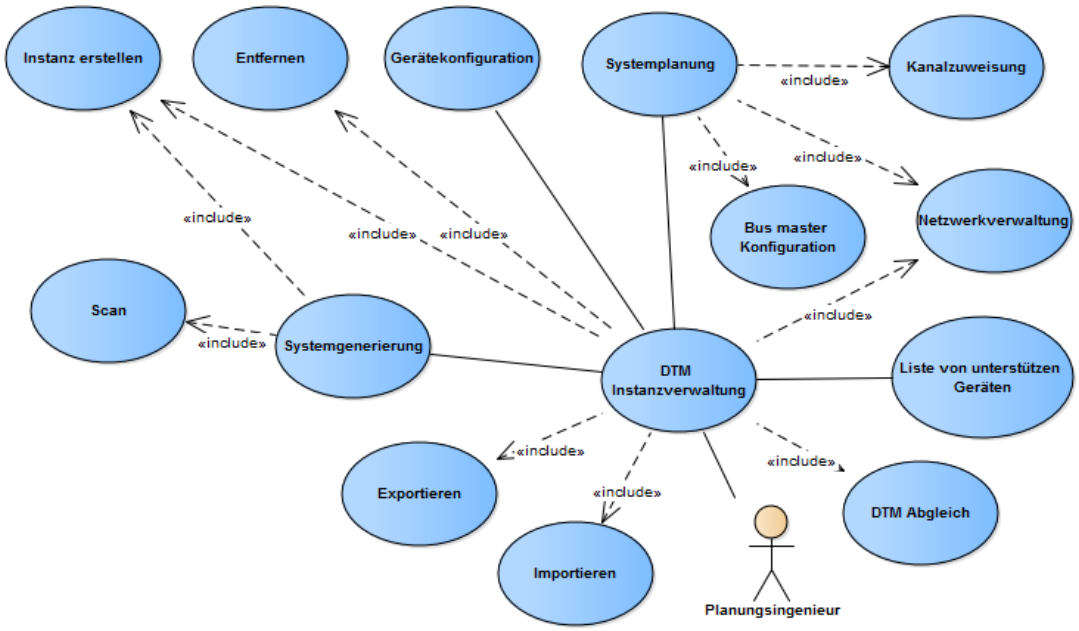

Abbildung B.5.: Akteur Planungsingenieur und dessen zugeordnete Anwendungsfälle (basiert auf Informationen aus [46])

Tabelle B.2.: Detailbeschreibung aller Anwendungsfälle von Field Device Tool 2 (basiert auf Informationen aus [46])

Anwendungsfall	**Beinhalteter Anwendungsfall**	**Beschreibung**
Onlinesicht		Bietet Operationen für die Betrachtung von Geräteparametern und Gerätestatus
	Parameter laden	Lädt Parameter vom Gerät für die Betrachtung sowie zur Dokumentation (Drucken)
	Online Status	Betrachtung und Analysierung des aktuellen Gerätestatus
	Online Trend	Generierung und Beobachtung von dynamischen Onlinewerten
	Plausibilitätsprüfung	Prüft jedes mal, wenn Geräteparameter oder Konfigurationsdaten geändert wurden, ob diese plausibel sind. Solange die Daten nicht plausibel sind, werden diese nicht in das Gerät geschrieben
	Identifizierung	Zeigt Identifikationsdaten für eine Geräteinstanz, zum Beispiel Geräteadresse oder die Firmware
	Scan	Empfängt eine Liste mit verfügbaren Geräten, hierbei kann die Liste von einem DTM oder von der Rahmenanwendung kommen

Fortsetzung auf nächster Seite

Tabelle B.2 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter Anwendungsfall**	**Beschreibung**
Berichtgenerierung		Mechanismen mit denen sich konfigurierbare Berichte erstellen lassen
Anlagenverwaltung		Rahmenanwendung bietet Funktionen zur Anlagenverwaltung an
Benutzerzugang		Eine Person, die einer bestimmten Nutzerrolle zugeordnet ist, verifiziert sich an der Rahmenanwendung um Zugang zu erlangen
	DTM spezifischer Zugang	Zugriff auf herstellerspezifische Informationen, wie zum Beispiel Produktionscodes oder Änderungsberichte
Audit Trail		Anzeigen und Löschen von Audit Trail Daten
	DTM spezifischer Audit Trail	Jeder DTM, kann einen eigenen Audit Trail unterstützen
	Rahmenanwendung spezifischer Audit Trail	Rahmenanwendung kann einen systemweiten, globalen Audit Trail implementieren
Einstellen von SetValue		Einstellen der SetValues bei einem Gerät, zum Beispiel Positioner oder Controllers

Fortsetzung auf nächster Seite

Tabelle B.2 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter Anwendungsfall**	**Beschreibung**
Archiv		Zugriff auf das Archiv der Rahmenanwendung zum Zwecke der Betrachtung und Datenanalyse
	Historischer Trend	Visualisierung von historischen Daten
	Historische Datenanalyse	Erweiterte Operationen zur Analysierung von historischen Daten
Simulation		Simuliert das Verhalten eines Geräts
Offline Operationen		Operationen, die keine bestehende Verbindung zum Gerät benötigen (Ausnahme: Für Up- und Download wird eine temporäre Verbindung benötigt)
	Plausibilitätsprüfung	Prüft jedes mal, wenn Geräteparameter geändert wurden, ob diese plausibel sind. Solange die Daten nicht plausibel sind, können diese nicht in die Datenbank oder in das Gerät geschrieben werden
	Offline Parametrisierung	Änderungen an den Parameterwerten werden in der Datenbank der Rahmenanwendung gespeichert

Fortsetzung auf nächster Seite

Tabelle B.2 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter Anwendungsfall**	**Beschreibung**
	Datenvergleich	Vergleich von Offline Parametersätzen mit Parametersätzen anderer Instanzen, hierbei können Datensätze editierbar sein und können von einem Datensatz in einen anderen Datensatz transferiert werden
Gerätereperatur		Operationen, die dann aufgerufen werden, wenn ein Gerät repariert oder geändert werden soll
DTM Instanzaufrüstung und Ersetzung		Operationen die einen DTM aufrüsten oder ersetzen und immer dann durchgeführt werden, wenn ein neuer DTM sich von einem vorhergehenden unterscheidet
	Aufrüstung	Operationen, die ausgeführt werden, wenn die zu vergleichenden DTMs ein kompatibles Datensatz Format haben
	Ersetzung	Operationen, die ausgeführt werden, wenn die zu vergleichenden DTMs kein kompatibles Datensatz Format haben
Online Operationen		Operationen, die direkt auf oder mit dem Gerät ausgeführt werden
	Onlinefunktionen	Bietet alle Prozeduren an, die eine direkte Kommunikation mit dem Gerät benötigen

Fortsetzung auf nächster Seite

Tabelle B.2 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter Anwendungsfall**	**Beschreibung**
	Online Parametrisierung	Nachdem alle Parameter vom Gerät gelesen werden, werden mögliche Änderungen an den gelesenen Parametern immer direkt auch auf dem Gerät geändert
	Datenvergleich	Vergleich von persistenten Daten (offline) mit den Gerätedaten, hierbei können Datensätze editierbar sein und können von einem Datensatz in einen anderen Datensatz transferiert werden
	Kalibrierung	Kalibrierungsprozeduren um Eingaben zu kalibrieren (zum Beispiel Pressure transmitters)
	Plausibilitätsprüfung	Prüft jedes mal, wenn Geräteparameter oder Konfigurationsdaten geändert wurden, ob diese plausibel sind. Solange die Daten nicht plausibel sind, werden diese nicht in das Gerät geschrieben
	Upload	Liest den kompletten Parametersatz vom Gerät in die DTM Instanzdaten, anschließend können die Daten in der Datenbank der Rahmenanwendung gespeichert werden

Fortsetzung auf nächster Seite

Tabelle B.2 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter Anwendungsfall**	**Beschreibung**
	Download	Schreibt den kompletten Parametersatz der DTM Instanzdaten in das Gerät
Massenoperationen		Kann eine Gruppe von Geräten mit einer Operation verwalten, zum Beispiel Up- und Download
	Upload	Liest die gesamte Parametrisierung von den Geräten der Gruppe in die Rahmenanwendungsdatenbank
	Download	Schreibt die gesamte Parametrisierung von der Rahmenanwendungsdatenbank in die Gruppe von Geräten
DTM Instanzverwaltung		Verwaltung einer Geräteinstanz in der Rahmenanwendung, zum Beispiel Instanziieren, Importieren, Exportieren, Entfernen
	Instanz erstellen	Neue Geräteinstanz kann erstellt und in das Projekt integriert werden

Fortsetzung auf nächster Seite

Tabelle B.2 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter Anwendungsfall**	**Beschreibung**
	Importieren	Parametersatz wird instanziiert durch Importierten der Daten aus der Datenbank (zum Beispiel aus einer Vorlagedatenbank (Template)) oder durch Kopieren des Parametersatzes von einem bereits instanziierten und verifizieren Gerät
	Exportieren	Um Daten von einer Geräteinstanz zu einer anderen zu kopieren, können die Daten einer Geräteinstanz exportiert werden
	Entfernen	Entfernen der Geräteinstanz
Gerätekonfiguration		Konfiguration eines komplexen Geräts, welches Konfiguration von Modulen inkludieren kann
Systemgenerierung		Operationen um alle notwendigen Aktionen durchzuführen um eine Topologie basierend auf dem Resultat eines Scans zu erzeugen
	Scan	Empfängt eine Liste mit verfügbaren Geräten, hierbei kann die Liste von einem DTM oder von der Rahmenanwendung kommen

Fortsetzung auf nächster Seite

Tabelle B.2 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter Anwendungsfall**	**Beschreibung**
	Netzwerkverwaltung	Verwaltung des Netzwerks, zum Beispiel Adresssetzung für Geräte
	DTM Abgleichung	DTM finden, welcher am besten passt, ausgehend von den Informationen, die beim Scan empfangen wurden
	Instanz erstellen	Neue Geräteinstanz kann erstellt und in das Projekt integriert werden
Systemplanung		Alle notwendigen Operationen für das Editieren der Topologie Informationen
	Netzwerkverwaltung	Verwaltung des Netzwerks, zum Beispiel Adresssetzung für Geräte
	Bus master Konfiguration	Konfiguration der bus master DTMs
	Kanalzuweisung	Editieren der FDT Kanalzuweisungen
Liste von unterstützten Geräten		Liste mit allen unterstützten Geräten innerhalb eines Projekts können angeschaut und editiert werden

B.3. Anwendungsfälle OPC UA Informationsmodell für FDT

In diesem Unterkapitel werden die aus dem Standard extrahierten Anwendungsfälle des *OPC UA Informationsmodell für FDT* aufgelistet und beschrieben.

Tabelle B.3.: Anwendungsfälle des OPC UA Informationsmodell für FDT (basiert auf Informationen aus [41])

Anwendungsfall	**Beinhalteter / Spezialisierter Anwendungsfall**	**Beschreibung**
Topologie auflisten		Ein OPC UA Client zeigt dem Benutzer, die Topologie in einer hierarchischen Struktur
Gerät identifizieren		Der Benutzer erhält Informationen über ein physisches Gerät, dass er in der Topologie ausgewählt hat
Liste verfügbarer Geräteparameter bekommen		
	Geräteparameter durchsuchen	Zeigt dem Nutzer die Blocknamen an, die das physikalische Gerät hat sowie die Parameternamen, welche die Blöcke beinhalten. Gibt es keine Blöcke im physikalischen Gerät, werden die Parameternamen angezeigt, die das Gerät hat
	Attribute eines Geräteparameters bekommen	Dem Benutzer werden die Parameternamen sowie deren Ingenieurseinheiten und Werte (sofern der Parameter lesbar ist) angezeigt. Der Parameter ist ggf. Beschreibbar

Fortsetzung auf nächster Seite

Tabelle B.3 – *Fortgeführt von vorangegangener Seite*

Anwendungsfall	**Beinhalteter / Spezialisierter Anwendungsfall**	**Beschreibung**
Gerätestatus		Status des Geräts oder Status des Ergebnisses der Diagnose eines Geräts
Gerätediagnose		Diagnose Informationen eines physikalischen Geräts
Parameter lesen		
	Offline Parameter lesen	Die letzten bei einem DTM gesetzten Parameter werden gelesen
	Online Parameter lesen	Die aktuellen Werte vom Gerät werden über den DTM gelesen
Geräteparameter schreiben		Konfiguration oder Änderung des Wertes eines physikalischen Geräts
Audit Trail		Zentralisiertes System ermöglicht die Aufzeichnung von Anwendern und deren Änderungen an einem physikalischen Gerät innerhalb einer bestimmten Zeitspanne und generiert für definierte Events Nachrichten, die zuvor von einer UA Anwendung abonniert werden müssen

C. Fehlerlisten der analysierten Stacks

Die Fehlerlisten zeigen die im Zusammenhang mit der plattformunabhängigen Entwicklung, Fehler verursachenden Quellcodestellen der analysierten Stacks. Hierbei wurden die Stacks mit der in Kapitel 4.2 beschriebenen Quellcodeverteilungsstrategie *Gemeinsames Projekt* analysiert. Aufgrund der hohen Seitenzahl der Dokumente liegen diese auf der beiliegenden DVD oder können über die genannten Webadressen als Portable Document Format (PDF) über HTTP aufgerufen werden. Auf beiliegender DVD können zusätzlich die für die Analyse angelegten Programmquellcodeprojekte betrachtet werden.

C.1. Fehlerliste für die Einbindung des UA .NET Stack

Die folgende Fehlerliste zeigt die Fehler verursachenden Quellcodestellen aller analysierten Plattformen des *.NET UA Stack* in einem Gesamtdokument. Für jede analysierte Plattform ist zusätzlich ein separates Dokument aufgeführt, dass nur die Fehler verursachenden Quellcodestellen enthält, die bei der jeweiligen Plattform auftreten.

Summierte Fehlerliste aller Plattformen Die Fehlerliste enthält alle Fehler verursachenden Quellcodestellen für die Plattformen Android, iOS, Universal Windows Plattform, Windows Phone 8.1 sowie Windows 8.1. Die folgenden Informationen werden dabei von dieser Liste repräsentiert: Visual Studio Compiler Error Code, Beschreibung des Fehlers, Plattform oder Plattformen auf dem der Fehler auftritt, Datei, in welcher der Fehler auftritt sowie die Zeile innerhalb der Datei, in welcher der Fehler auftritt.

Auf beiliegender DVD unter folgendem Pfad zu finden: UA .NET Stack/analyse-errorlog.pdf
Via HTTP unter folgender Webadresse zu finden: http://ts.digital/bsc1

Fehlerliste für Android Die Fehlerliste enthält alle Fehler verursachenden Quellcodestellen für die Plattform Android. Die folgenden Informationen werden dabei von dieser Liste repräsentiert: Visual Studio Compiler Error Code, Beschreibung des Fehlers, Datei, in welcher der Fehler auftritt sowie die Zeile innerhalb der Datei, in welcher der Fehler auftritt.

Auf beiliegender DVD unter folgendem Pfad zu finden: UA .NET Stack/analyse-errorlog-android.pdf
Via HTTP unter folgender Webadresse zu finden: http://ts.digital/bsc2

Fehlerliste für iOS Die Fehlerliste enthält alle Fehler verursachenden Quellcodestellen für die Plattform iOS. Die folgenden Informationen werden dabei von dieser Liste repräsentiert: Visual Studio Compiler Error Code, Beschreibung des Fehlers, Datei, in welcher der Fehler auftritt sowie die Zeile innerhalb der Datei, in welcher der Fehler auftritt.

Auf beiliegender DVD unter folgendem Pfad zu finden: UA .NET Stack/analyse-errorlog-ios.pdf
Via HTTP unter folgender Webadresse zu finden: http://ts.digital/bsc3

Fehlerliste für Universal Windows Plattform Die Fehlerliste enthält alle Fehler verursachenden Quellcodestellen für die Plattform Universal Windows Plattform. Die folgenden Informationen werden dabei von dieser Liste repräsentiert: Visual Studio Compiler Error Code, Beschreibung des Fehlers, Datei, in welcher der Fehler auftritt sowie die Zeile innerhalb der Datei, in welcher der Fehler auftritt.

Auf beiliegender DVD unter folgendem Pfad zu finden: UA .NET Stack/analyse-errorlog-uwp.pdf
Via HTTP unter folgender Webadresse zu finden: http://ts.digital/bsc4

Fehlerliste für Windows Phone 8.1 Die Fehlerliste enthält alle Fehler verursachenden Quellcodestellen für die Plattform Windows Phone 8.1. Die folgenden Informationen werden dabei von dieser Liste repräsentiert: Visual Studio Compiler Error Code, Beschreibung des Fehlers, Datei, in welcher der Fehler auftritt sowie die Zeile innerhalb der Datei, in welcher der Fehler auftritt.

Auf beiliegender DVD unter folgendem Pfad zu finden: UA .NET Stack/analyse-errorlog-winPhone8dot1.pdf
Via HTTP unter folgender Webadresse zu finden: http://ts.digital/bsc5

Fehlerliste für Windows 8.1 Die Fehlerliste enthält alle Fehler verursachenden Quellcodestellen für die Plattform Windows 8.1. Die folgenden Informationen werden dabei von dieser Liste repräsentiert: Visual Studio Compiler Error Code, Beschreibung des Fehlers, Datei, in welcher der Fehler auftritt sowie die Zeile innerhalb der Datei, in welcher der Fehler auftritt.

Auf beiliegender DVD unter folgendem Pfad zu finden: UA .NET Stack/analyse-errorlog-windows8dot1.pdf
Via HTTP unter folgender Webadresse zu finden: http://ts.digital/bsc6

C.2. Fehlerliste für die Einbindung des UA .NET UWP Stacks

Die folgende Fehlerliste zeigt die Fehler verursachenden Quellcodestellen aller analysierten Plattformen des *.NET UWP UA Stack* in einem Gesamtdokument. Für die analysierten Plattformen Android, Windows Phone 8.1 und Windows 8.1 ist zusätzlich ein separates Dokument aufgeführt, dass nur die Fehler verursachenden Quellcodestellen enthält, die bei der jeweiligen Plattform auftreten. Für die Plattformen UWP und iOS konnten keine Quellcodestellen ermittelt werden, welche die von der Plattform unabhängige Anwendungsentwicklung beeinträchtigen.

Summierte Fehlerliste aller Plattformen Die Fehlerliste enthält alle Fehler verursachenden Quellcodestellen für die Plattformen Android, Windows Phone 8.1 sowie Windows 8.1. Die folgenden Informationen werden dabei von dieser Liste repräsentiert: Visual Studio Compiler Error Code, Beschreibung des Fehlers, Plattform oder Plattformen auf dem der Fehler auftritt, Datei, in welcher der Fehler auftritt sowie die Zeile innerhalb der Datei, in welcher der Fehler auftritt.

Auf beiliegender DVD unter folgendem Pfad zu finden: UA .NET UWP Stack/analyse-errorlog.pdf
Via HTTP unter folgender Webadresse zu finden: http://ts.digital/bsc7

Fehlerliste für Android Die Fehlerliste enthält alle Fehler verursachenden Quellcodestellen für die Plattform Android. Die folgenden Informationen werden dabei von dieser Liste repräsentiert: Visual Studio Compiler Error Code, Beschreibung des Fehlers, Datei, in welcher der Fehler auftritt sowie die Zeile innerhalb der Datei, in welcher der Fehler auftritt.

Auf beiliegender DVD unter folgendem Pfad zu finden: UA .NET UWP Stack/analyse-errorlog-android.pdf
Via HTTP unter folgender Webadresse zu finden: http://ts.digital/bsc8

Fehlerliste für Windows Phone 8.1 Die Fehlerliste enthält alle Fehler verursachenden Quellcodestellen für die Plattform Windows Phone 8.1. Die folgenden Informationen werden dabei von dieser Liste repräsentiert: Visual Studio Compiler Error Code, Beschreibung des Fehlers, Datei, in welcher der Fehler auftritt sowie die Zeile innerhalb der Datei, in welcher der Fehler auftritt.

Auf beiliegender DVD unter folgendem Pfad zu finden: UA .NET UWP Stack/analyse-errorlog-winPhone8dot1.pdf
Via HTTP unter folgender Webadresse zu finden: http://ts.digital/bsc9

Fehlerliste für Windows 8.1 Die Fehlerliste enthält alle Fehler verursachenden Quellcodestellen für die Plattform Windows 8.1. Die folgenden Informationen werden dabei von dieser Liste repräsentiert: Visual Studio Compiler Error Code, Beschreibung des Fehlers, Datei, in welcher der Fehler auftritt sowie die Zeile innerhalb der Datei, in welcher der Fehler auftritt.

Auf beiliegender DVD unter folgendem Pfad zu finden: UA .NET UWP Stack/analyse-errorlog-win8dot1.pdf
Via HTTP unter folgender Webadresse zu finden: http://ts.digital/bsc10

Literaturverzeichnis

[1] W. Mahnke and S.-H. Leitner, “System Architecture,” in *OPC Unified Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, Kapitel 9, Seiten 265–282. [Online]. Verfügbar: http://ts.digital/switch

[2] W. Ertel, “Einführung,” in *Grundkurs Künstliche Intelligenz*. Wiesbaden: Springer Fachmedien Wiesbaden, 2013, Kapitel 1, Seiten 1–17. [Online]. Verfügbar: http://ts.digital/ertel

[3] C. Schawel and F. Billing, “Wertschöpfungskette,” in *Top 100 Management Tools*. Wiesbaden: Gabler Verlag, 2014, Seiten 279–281. [Online]. Verfügbar: http://ts.digital/schawel

[4] M. Hartmann, “Technologie,” in *Handbuch Cultural Studies und Medienanalyse*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, Kapitel 2, Seiten 351–360. [Online]. Verfügbar: http://ts.digital/hartmann

[5] C. Petzold, *Creating Mobile Apps with Xamarin. Forms*, 2016. [Online]. Verfügbar: http://ts.digital/petzold

[6] Xamarin Inc., “Mobile Application Development to Build Apps in C# - Xamarin,” Seiten 1–8. [Online]. Verfügbar: http://ts.digital/xamarininc2016 (zuletzt abgerufen: 2016-07-07)

[7] V. P. Andelfinger and T. Hänisch, “Einleitung,” in *Internet der Dinge*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, Kapitel 1, Seiten 1–8. [Online]. Verfügbar: http://ts.digital/andelfinger

[8] T. Malthus, *An Essay on the Principle of Population*. London: Electronic Scholarly Publishing Project, 1798. [Online]. Verfügbar: http://ts.digital/malthus

[9] B. Schäfers, “Die Veränderung der Lebensgrundlagen durch die Industrielle Revolution,” in *Sozialgeschichte der Soziologie*. Wiesbaden: Springer Fachmedien Wiesbaden, 2016, Kapitel 2, Seiten 15–26. [Online]. Verfügbar: http://ts.digital/schafers

[10] H. Kagermann, W. Wahlster, and J. Helbig, “Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0,” Seiten 1–112, 2013. [Online]. Verfügbar: http://ts.digital/henningkargermannwolfgangwahlster

[11] R. Anderl, “Was ist Industrie 4.0?” Seiten 1–13, 2016. [Online]. Verfügbar: http://ts.digital/reineranderl (zuletzt abgerufen: 2016-07-25)

[12] C. Pinnow and S. Schäfer, *Industrie 4.0 - Grundlagen und Anwendungen: Branchentreff der Berliner Wissenschaft und Industrie*, D. e.V., Ed. Beuth Verlag, 2015. [Online]. Verfügbar: http://ts.digital/pinnow

[13] S. Hoppe, “Standardisierte horizontale und vertikale Kommunikation,” in *Handbuch Industrie 4.0*, B. Vogel-Heuser, T. Bauernhansl, and M. ten Hompel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, Seiten 1–20. [Online]. Verfügbar: http://ts.digital/hoppe

[14] Bundesministerium für Bildung und Forschung, “Zukunftsprojekt Industrie 4.0,” Seiten 1–7. [Online]. Verfügbar: http://ts.digital/bmbf (zuletzt abgerufen: 2016-07-25)

[15] V. Koch, R. Geissbauer, S. Kuge, and S. Schrauf, “Industrie 4.0 - Chancen und Herausforderungen der vierten industriellen Revolution,” Strategy&, PwC, Tech. Rep., 2014. [Online]. Verfügbar: http://ts.digital/koch

[16] D. Siepmann and N. Graef, “Industrie 4.0 – Grundlagen und Gesamtzusammenhang,” in *Einführung und Umsetzung von Industrie 4.0*, A. Roth, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, Kapitel 2, Seiten 17–82. [Online]. Verfügbar: http://ts.digital/siepmann

[17] W. Wahlster, “Technologische Grundlagen der Industrie 4.0,” Seiten 1–32, 2016. [Online]. Verfügbar: http://ts.digital/wahlster (zuletzt abgerufen: 2016-07-25)

[18] M. Schindler, S. Magerstedt, J. Regtmeier, T. Kaufmann, L. Sören Bochmann, L. Gehrke, N. Gehrke, C. Mertens, O. Seiss, A. Hermann, L. Longhitano, T. Paulus, M. Hauske, D. Hajizadeh, C. Kreibich, S. Meinzer, S. Augustine, and J. Sarah Geffers, “Industrie 4.0 –Use Cases aus Forschung und Unternehmenspraxis,” in *Einführung und Umsetzung von Industrie 4.0*, A. Roth, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, Kapitel 4, Seiten 133–246. [Online]. Verfügbar: http://ts.digital/schindler

[19] Bundesministerium für Wirtschaft und Energie, “Industrie 4.0 und Digitale Wirtschaft,” Berlin, Tech. Rep., 2015. [Online]. Verfügbar: http://ts.digital/bwwi

[20] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (GMA), “Thesen und Handlungsfelder - Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation,” Verein Deutscher Ingenieure e.V., Tech. Rep., 2013. [Online]. Verfügbar: http://ts.digital/vereindeutscheringenieureev

[21] D. Zühlke, “Interoperabilität in der Smart Factory - Teil 1,” Seiten 1–5, 2016. [Online]. Verfügbar: http://ts.digital/zuhlke (zuletzt abgerufen: 2016-07-25)

[22] P. Mertens, *Integrierte Informationsverarbeitung 1*. Gabler Verlag, 2013. [Online]. Verfügbar: http://ts.digital/mertens

[23] P. Schubert and A. Winkelmann, “Vorlesung Einführung in die Wirtschafts- und Verwaltungsinformatik,” 2014. [Online]. Verfügbar: http://ts.digital/schubert

[24] Bundesministerium für Wirtschaft und Technologie, “Machine-to-Machine- Kommunikation – eine Chance für die deutsche Industrie,” Berlin, Tech. Rep., 2012. [Online]. Verfügbar: http://ts.digital/bundesministeriumfurwirtschaftundtechnologie

[25] A. Bildstein and J. Seidelmann, “Industrie 4.0-Readiness: Migration zur Industrie 4.0-Fertigung,” in *Industrie 4.0 in Produktion, Automatisierung und Logistik*, T. Bauernhansl, M. ten Hompel, and B. Vogel-Heuser, Eds. Wiesbaden: Springer Fachmedien Wiesbaden, 2014, Kapitel 30, Seiten 581–597. [Online]. Verfügbar: http://ts.digital/bildstein

[26] V. Bouse and G. Süss, “Durchgängiger Datenaustausch,” *open automation*, Seiten 1–3, 2014. [Online]. Verfügbar: http://ts.digital/bouse2014

[27] OPC Foundation, “Members,” Seiten 1–4. [Online]. Verfügbar: http://ts.digital/opcfoundation2016c (zuletzt abgerufen: 2016-07-25)

[28] DUDEN, “Plattform,” Bibliographisches Institut GmbH, Tech. Rep. [Online]. Verfügbar: http://ts.digital/dudenpltf

[29] V. Bouse and A. Röck, “Produktions- und IT-Welt mit OPC UA verbinden,” *etz elektrotechnik & automation*, Seiten 4–6, 2015. [Online]. Verfügbar: http://ts.digital/bouse

[30] W. Mahnke and S.-H. Leitner, “Standard Information Models,” in *OPC Unified Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, Kapitel 4, Seiten 107–123. [Online]. Verfügbar: http://ts.digital/mahnke2009b

[31] ——, “Introduction,” in *OPC Unified Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, Kapitel 1, Seiten 1–17. [Online]. Verfügbar: http://ts.digital/mahnke2009c

[32] OPC Foundation, “OPC Unified Architecture - Wegbereiter der 4. industriellen (R)Evolution,” Tech. Rep. [Online]. Verfügbar: http://ts.digital/opcfoundation

[33] S. Kleuker, “Anforderungsanalyse,” in *Grundkurs Software-Engineering mit UML*. Wiesbaden: Springer Fachmedien Wiesbaden, 2013, Kapitel 4, Seiten 55–92. [Online]. Verfügbar: http://ts.digital/kleuker

[34] P. Schönsleben, “Systems Engineering und Projektmanagement,” in *Integrales Logistikmanagement*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, Kapitel 19, Seiten 763–787. [Online]. Verfügbar: http://ts.digital/schonsleben

[35] V. Van Tan, D. S. Yoo, and M. J. Yi, “Device Integration Approach to OPC UA-Based Process Automation Systems with FDT/DTM and EDDL,” in *Lecture Notes in Computer Science*, D.-S. Huang, K.-H. Jo, H.-H. Lee, H.-J. Kang, and V. Bevilacqua, Eds., vol. 5755. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, Seiten 1001–1012. [Online]. Verfügbar: http://ts.digital/vantan

[36] W. Mahnke and S.-H. Leitner, “Services,” in *OPC Unified Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, Kapitel 5, Seiten 125–190. [Online]. Verfügbar: http://ts.digital/mahnke

[37] OPC Foundation, “OPC Unified Architecture Specification - Part 4: Services,” Tech. Rep., 2015. [Online]. Verfügbar: http://ts.digital/unified2015

[38] ——, “OPC Unified Architecture Specification - Part 12: Discovery,” Tech. Rep., 2015. [Online]. Verfügbar: http://ts.digital/unified

[39] M. Damm, “OPC UA Discovery,” Seiten 1–31, 2014. [Online]. Verfügbar: http://ts.digital/damm

[40] W. Mahnke and S.-H. Leitner, “Information Modeling: Concepts,” in *OPC Unified Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, Kapitel 2, Seiten 19–84. [Online]. Verfügbar: http://ts.digital/mahnke2009a

[41] OPC Foundation and FDT Group, “FDT Group and OPC Foundation: OPC UA Information Model for FDT (Draft for Member Review 0.17),” Tech. Rep., 2016.

[42] M. Brill and F. Schmid, “Mit FDT und OPC UA in Richtung Industrie 4.0,” *SPS Magazin*, Seiten 44–45, 2016. [Online]. Verfügbar: http://ts.digital/brill2016

[43] D. Grossmann, K. Bender, and B. Danzer, “OPC UA based Field Device Integration,” *SICE Annual Conference*, Seiten 933–938, 2008. [Online]. Verfügbar: http://ts.digital/grossmann

[44] FDT Group, “FDT technology, what is it?” Seiten 1–2. [Online]. Verfügbar: http://ts.digital/unknown (zuletzt abgerufen: 2016-07-25)

[45] H. Yuan and F. Liu, “Research on OPC UA based on FDT/DTM and EDDL,” *Digital Manufacturing and Automation (ICDMA)*, Seiten 992–995, 2011. [Online]. Verfügbar: http://ts.digital/yuan

[46] FDT Group, “FDT2.0 Technical Specification,” Jodoigne, Tech. Rep., 2012. [Online]. Verfügbar: http://ts.digital/specification1http://ts.digital/specification2

[47] M. Brill, “Geräteintegration mit FDT and OPC UA - Ein Baustein für Industrie 4.0,” *Forum Industrie 4.0*, Seiten 1–27, 2015. [Online]. Verfügbar: http://ts.digital/brill

[48] U. Topp, “FDT device model for OPC UA,” Tech. Rep., 2008.

[49] FDT Group, “FDT: Eine Technische Beschreibung,” Tech. Rep.

[50] S. R. Humayoun, S. Ehrhart, and A. Ebert, “Developing Mobile Apps Using Cross-Platform Frameworks: A Case Study,” in *Lecture Notes in Computer Science*, M. Kurosu, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, Seiten 371–380. [Online]. Verfügbar: http://ts.digital/humayoun

[51] IDC Research, “Smartphone OS Market Share, 2015 Q2,” Seiten 1–2, 2015. [Online]. Verfügbar: http://ts.digital/n2015 (zuletzt abgerufen: 2016-07-25)

[52] R. Ghatol and Y. Patel, “Understanding Cross-Platform Mobile Application Development,” in *Beginning PhoneGap*, 2012, Kapitel 1, Seiten 1–16. [Online]. Verfügbar: http://ts.digital/ghatol

[53] Gartner, “Gartner Says Smartphone Sales Surpassed One Billion Units in 2014,” Seiten 1–7, 2015. [Online]. Verfügbar: http://ts.digital/gartner (zuletzt abgerufen: 2016-07-25)

[54] H. Heitkötter, T. A. Majchrzak, B. Ruland, and T. Weber, “Comparison of Mobile Web Frameworks,” in *Lecture Notes in Business Information Processing*, K.-H. Krempels and A. Stocker, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, Seiten 119–137. [Online]. Verfügbar: http://ts.digital/heitkotter

[55] M. Korf and E. Oksman, “Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options,” Seiten 1–7, 2016. [Online]. Verfügbar: http://ts.digital/korf

[56] D. Hermes, “Mobile Development Using Xamarin,” in *Xamarin Mobile Application Development*. Apress, 2015, Kapitel 1, Seiten 1–8. [Online]. Verfügbar: http://ts.digital/hermes

[57] Xamarin Inc., “Platform Features,” Tech. Rep. [Online]. Verfügbar: http://ts.digital/xamarininca

[58] ——, “Sharing Code Options,” Seiten 1–5. [Online]. Verfügbar: http://ts.digital/xamarin2013a (zuletzt abgerufen: 2016-07-12)

[59] ——, “Introduction to Portable Class Libraries,” Seiten 1–13. [Online]. Verfügbar: http://ts.digital/xamarininc2016b (zuletzt abgerufen: 2016-07-14)

[60] ——, “Shared Projects,” Seiten 1–7. [Online]. Verfügbar: http://ts.digital/xamarininc2016a (zuletzt abgerufen: 2016-07-14)

[61] OPC Foundation, “OPC Unified Architecture Specification,” Tech. Rep. [Online]. Verfügbar: http://ts.digital/bernhard2010a

[62] C. Moser, “Nutzerforschung,” in *User Experience Design*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, Kapitel 4, Seiten 55–84. [Online]. Verfügbar: http://ts.digital/moser

[63] Bundesagentur für Arbeit and Statistik/Arbeitsmarktberichterstattung, “Der Arbeitsmarkt in Deutschland Frauen und Männer am Arbeitsmarkt 2014,” Nürnberg, Tech. Rep., 2015. [Online]. Verfügbar: http://ts.digital/bundesagenturfurarbeit

[64] C. Wingerter, “Beschäftigungsfelder von Männern und Frauen,” Seiten 1–2, 2008. [Online]. Verfügbar: http://ts.digital/wingerter (zuletzt abgerufen: 2016-05-13)

[65] B. Frost, “Atomic Design Methodology,” Seiten 1–30, 2016. [Online]. Verfügbar: http://ts.digital/frost (zuletzt abgerufen: 2016-07-11)

[66] B. Fling, “Mobile Information Architecture,” in *Mobile Design and Development*. O’Reilly Media, Inc., 2009, Kapitel 7. [Online]. Verfügbar: http://ts.digital/fling

[67] C. Moser, “Informationsarchitektur,” in *User Experience Design*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, Kapitel 6, Seiten 105–120. [Online]. Verfügbar: http://ts.digital/moser2013c

[68] J. Nichols, “Automatically Generating High-Quality User Interfaces for Appliances,” Computer Science, Carnegie Mellon University, 2006. [Online]. Verfügbar: http://ts.digital/nichols

[69] K. Z. Gajos, “Automatically Generating Personalized User Interfaces,” Ph.D. dissertation, University of Washington, 2008. [Online]. Verfügbar: http://ts.digital/gajos

[70] C. Moser, “Visual Design,” in *User Experience Design*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, Kapitel 9, Seiten 181–218. [Online]. Verfügbar: http://ts.digital/klemmer

[71] Scott Design Inc, “Color Preferences,” Seiten 1–3. [Online]. Verfügbar: http://ts.digital/scottdesigninc (zuletzt abgerufen: 2016-06-21)

[72] ——, “What’s your favorite color? [Infographic],” Seiten 1–6. [Online]. Verfügbar: http://ts.digital/scottdesigninc2016a (zuletzt abgerufen: 2016-06-21)

[73] J. Olesen, “Blue Color Meaning – The Color Blue,” Seiten 1–3. [Online]. Verfügbar: http://ts.digital/olesen (zuletzt abgerufen: 2016-06-21)

[74] T. Krantz, *Designe deine eigene Marke in nur 3 Stunden*, 2015. [Online]. Verfügbar: http://ts.digital/krantz

[75] E. Schmich, “Dieses GIF zeigt euch, wie Farbenblinde sehen,” Seiten 1–2, 2015. [Online]. Verfügbar: http://ts.digital/schmich (zuletzt abgerufen: 2016-07-25)

[76] J. Olesen, “Green Color Meaning – The Color Green,” Seiten 1–3. [Online]. Verfügbar: http://ts.digital/olesen2016 (zuletzt abgerufen: 2016-07-25)

[77] Uxmovement, “Why You Should Never Pair Green and Red on the Web,” Seiten 1–5, 2010. [Online]. Verfügbar: http://ts.digital/uxmovement (zuletzt abgerufen: 2016-07-25)

[78] J. Olesen, “Orange Color Meaning – The Color Orange,” Seiten 1–3. [Online]. Verfügbar: http://ts.digital/articlecolor (zuletzt abgerufen: 2016-07-25)

[79] OPC Foundation, “Unified Architecture,” Seite 1, 2015. [Online]. Verfügbar: http://ts.digital/bernhard (zuletzt abgerufen: 2016-07-22)

[80] ——, “OPC Unified Architecture Specification - Part 6: Mappings,” Tech. Rep., 2015. [Online]. Verfügbar: http://ts.digital/opcfoundation2015a

[81] Unified Automation, “.NET Based OPC UA Client & Server SDK (Bundle),” Seiten 1–2. [Online]. Verfügbar: http://ts.digital/unifiedautomation2016 (zuletzt abgerufen: 2016-07-22)

[82] L. Hohmann, *Beyond Software Architecture: Creating and Sustaining Winning Solutions*, 2003. [Online]. Verfügbar: http://ts.digital/hohmann2013

[83] B. Oestereich, *Analyse und Design mit der UML 2.1 – Objektorientierte Softwareentwicklung*. Oldenbourg, 2006. [Online]. Verfügbar: http://ts.digital/oestereich

[84] V. Gaudioso, “MVVM: Model-View-ViewModel,” in *Foundation Expression Blend 4 with Silverlight*. Berkeley, CA: Apress, 2010, Kapitel 15, Seiten 341–367. [Online]. Verfügbar: http://ts.digital/gaudioso

[85] Autofac, “Autofac,” Tech. Rep., 2013. [Online]. Verfügbar: http://ts.digital/autofac

[86] Xamarin Inc., “Working with a Local Database,” Seiten 1–12. [Online]. Verfügbar: http://ts.digital/xamarininc (zuletzt abgerufen: 2016-07-26)

[87] TwinCoders, “SQLite-Net Extensions,” Seiten 1–24. [Online]. Verfügbar: http://ts.digital/twincoders (zuletzt abgerufen: 2016-07-26)

[88] Red Hat, “Drools,” Tech. Rep., 2016. [Online]. Verfügbar: http://ts.digital/redhat2016

[89] G. Gappmeier, “OPC-UA-Architektur,” Seiten 1–2, 2006. [Online]. Verfügbar: http://ts.digital/gappmeier (zuletzt abgerufen: 2016-07-25)

[90] Softing Industrial Automation and A. Knoll, “OPC UA als Schlüssel zur Industrie 4.0,” *Markt&Technik*, Seiten 1–3, 2013. [Online]. Verfügbar: http://ts.digital/softingindustrialautomation

[91] Www.pd4pic.com, “GEOMETRY, HEXAGONS, LINES, MESH, NETWORK, POLYGONS,” Seiten 1–3. [Online]. Verfügbar: http://ts.digital/wwwpd4piccom (zuletzt abgerufen: 2016-07-25)

[92] OPC Foundation, “Markets & Collaboration,” Seiten 1–4. [Online]. Verfügbar: http://ts.digital/opcfoundation2016d (zuletzt abgerufen: 2016-07-25)

[93] Matrikon International, “Die Rolle von OPC Unified Architecture in der Industrie 4.0,” 2014. [Online]. Verfügbar: http://ts.digital/matrikoninternational

[94] OPC Foundation, “OPC Unified Architecture Specification - Part 1: Overview and Concepts,” Tech. Rep., 2015. [Online]. Verfügbar: http://ts.digital/opcfoundation2015