

Institut für Parallele und Verteilte Systeme

Abteilung Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Masterarbeit Nr. 93

Definition eines Datenhaltungskonzepts für Industrie 4.0 Anwendungen

Matthias Strljic

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer:	Dr. rer. nat. Frank Dürr
begonnen am:	01.04.2016
beendet am:	01.10.2016
CR-Klassifikation:	J.7, J.1

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.

Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.

Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.

Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.

Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Stuttgart, 27.09.2016

Declaration

I hereby declare that the work presented in this thesis is entirely my own.

I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.

Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.

The electronic copy is consistent with all submitted copies.

Signature:

Stuttgart, 27.09.2016

Definition eines Datenhaltungskonzepts für Industrie 4.0 Anwendungen

Masterarbeit



**Universität
Stuttgart**

Autor: **Matthias Strljic**
Matrikelnummer: 2675475

Betreuung: **Dr.-Ing. Jan Schlechtendahl,**
Dr. rer. nat. Frank Dürr

Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel

Universität Stuttgart

Studiengang: M. Sc. Softwaretechnik

Datum: 01.10.2016

Inhaltsverzeichnis

1	Einleitung.....	1
1.1	Motivation.....	1
1.2	Zielsetzung.....	1
2	Stand der Technik und Grundlagen	2
2.1	Maschineninteraktion.....	2
2.1.1	Maschinenprogrammierung.....	2
2.1.2	Maschinenkommunikation	6
2.2	Modellierung.....	10
2.2.1	Unified Modeling Language (UML).....	10
2.2.2	Business Process Model and Notation (BPMN).....	12
2.2.3	Computer - aided x (CAX)	15
2.2.4	Automation Mockup Language (AutomationML).....	17
2.3	Industrie 4.0 Wertschöpfungskette	21
2.4	RAMI 4.0	24
2.4.1	Referenzarchitektur.....	24
2.4.2	I4.0 Komponente.....	26
2.4.3	Verwaltungsschale.....	27
2.4.4	Servicearchitektur	30
3	Konzeption.....	32
3.1	Anforderungen an das Datenmodell.....	32
3.2	Herleitung des Datenmodells	33
3.3	Datenmodell.....	40
3.3.1	I4.0 Domäne	40
3.3.2	Servicebeschreibung.....	42
3.3.3	Prozessbeschreibung.....	45
3.3.4	Asset	47
3.3.5	I4.0 Komponente.....	50
4	Umsetzung.....	52
4.1	Wahl des Formates.....	52
4.2	Datenhaltung mit AutomationML.....	52
4.3	Realisierung mit AutomationML.....	58
4.3.1	Domäne	58
4.3.2	Servicebeschreibung.....	59
4.3.3	Prozessbeschreibung.....	61

4.3.4	Asset	63
4.3.5	I4.0 Komponente.....	65
4.4	Validierung anhand einer beispielhaften Umsetzung	65
4.5	Validierung der Anforderungen	75
5	Zusammenfassung und Ausblick.....	77
6	Literaturverzeichnis.....	80
7	Abbildungsverzeichnis.....	83
8	Tabellenverzeichnis	85

ABKÜRZUNGEN

OPC - Open Platform Communications

OPC UA - Open Platform Communications Unified Automation

OPC DA - Open Platform Communications Data Access

OPC AE - Open Platform Communications Alarms and Events

OPC HDA - Open Platform Communications History Data Access

UML - Unified Modeling Language

BPEL - Business Process Execution Language

BPMN - Business Process Management Notation

WS - Web Service

SOA - Service oriented Architecture

ADS - Automation Device Specification

I4.0 - Industrie 4.0

RAMI 4.0 - Referenzarchitekturmodell Industrie 4.0

CAX - Computer aided X

CAD - Computer aided design

CAM - Computer aided manufacturing

FRN - Funktionsresultatnachricht

FFN - Funktionsfehlernachricht

FAN - Funktionsaufrufnachricht

1 EINLEITUNG

Dieses Kapitel dient der Einführung in die Arbeit und soll klären, welche Motivation hinter dieser Arbeit steht und welche Ziele damit verfolgt werden. Hierfür wird in Kapitel 1.1 die Motivation aufgeführt und innerhalb von Kapitel 1.2 die Zielsetzung erläutert.

1.1 MOTIVATION

Das Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen (ISW) definiert den Begriff der „Digitalen Produktion“ als eines ihrer Leitthemen. Der Hintergrund hierbei besteht in der fortschreitenden Digitalisierung von Produktionslinien, der Entwicklung von neuen Produkten und deren Fertigungsplanung. Eine solche Digitalisierung bietet unter anderem die Verknüpfung von zuvor getrennten Bereichen der Produktion. Dies wiederum begünstigt, durch eine digitale Repräsentation der realen Welt, neue Wertschöpfungsmöglichkeiten entlang der gesamten Produktion. Das Verbinden von Informationen und Informationsflüssen spielt hierbei eine bedeutende Rolle. Der VDI GMA Fachausschuss 7.21 „Industrie 4.0“ [1] legte, mit den Wertschöpfungsketten von Industrie 4.0 (I4.0), den Informationsfluss zwischen den einzelnen Produktionsphasen offen. Die verschiedenen Produktionsphasen dieser Wertschöpfungsketten besitzen jeweils eigene Informationsquellen und Informationstypen. Diese Informationsquellen umfassen wiederum proprietäre Datenformate mit eigenen Datenmodellen, welche die Informationen in ihre digitale Repräsentation überführen. Daraus ergibt sich der entscheidende Konflikt, dass die Informationen mithilfe derer ein Mehrwert generiert werden soll, ohne ein einheitliches Datenformat, nicht gemeinsam innerhalb einer digitalen Umgebung verwendbar sind. Der VDI GMA Fachausschuss 7.21 strebt infolgedessen die Spezifikation eines solchen Datenformates mit der zugehörigen Datenhaltung an. Hierfür wurde bereits die I4.0 Begriffsdefinition [2] sowie ein Referenzarchitekturmodell [3] erstellt, welches die Rahmenbedingungen spezifiziert. Die Motivation dieser Arbeit ist die Unterstützung des VDI GMA Fachausschusses mithilfe eines Entwurfes, welcher ein mögliches Datenhaltungskonzept für Industrie 4.0 Anwendungen unter den Rahmenbedingungen des Referenzarchitekturmodells darstellt.

1.2 ZIELSETZUNG

Das Ziel dieser Arbeit ist ein Datenhaltungskonzept für I4.0 Anwendungen, welches dem VDI GMA Fachausschuss 7.21 als Entwurf vorgelegt werden soll. Dieses Konzept muss hierfür ein Datenmodell und dessen zugehörige Datenhaltung definieren. Für diese Definition müssen die innerhalb einer Wertschöpfungskette auftretenden Datenformate betrachtet werden. Ausgehend von den Informationen bezüglich dieser Datenformate soll ein Datenmodell erstellt werden, welches zudem die Rahmenbedingungen des Referenzarchitekturmodells erfüllt. Das Datenmodell muss in der Lage sein, Informationen aus den verschiedenen Datenformaten von Industrie 4.0 Anwendungen abstrahiert abzubilden. Für die Validierung soll dieses Datenmodell in ein maschinenlesbares Datenformat überführt werden. Das abschließende Ziel ist mithilfe dieses Datenformates das angestrebte Datenhaltungskonzept zu entwerfen. Dies soll umfassen wie die Daten einer I4.0 Anwendung durch das realisierte Datenmodell dargestellt und dessen Datenhaltung verwaltet werden. Die Definition dieses Datenhaltungskonzeptes steht unter dem Kontext von Industrie 4.0 und soll hierfür die Begriffsdefinitionen des VDI GMA Fachausschusses innerhalb dieser Arbeit und dessen resultierenden Datenhaltungskonzeptes verwenden.

2 STAND DER TECHNIK UND GRUNDLAGEN

Dieses Kapitel umfasst Grundlagen bezüglich der Interaktion mit Maschinen, der Modellierung von Softwaredaten sowie darin enthaltener Prozesse. Zusätzlich werden Informationen zum Stand der Technik bezüglich Industrie 4.0 zusammengetragen. Es werden hierfür das Referenzarchitekturmodell für die Industrie 4.0, die Wertschöpfungsketten sowie das sich in der Erarbeitung befindende Servicearchitekturmodell für Industrie 4.0 betrachtet. Bei der Beschreibung der folgenden Unterkapitel wurde ein besonderer Wert auf die dahinterliegenden Datenstrukturen sowie Modellierungsstrategien gelegt. Dies liegt daran, dass diese den größten Einfluss auf das angestrebte Datenhaltungskonzept besitzen und somit priorisiert wurden.

2.1 MASCHINENINTERAKTION

Die Maschineninteraktion wird in dieser Arbeit auf zwei Aspekte der digitalen Interaktion mit Fertigungsmaschinen beschränkt. Es handelt sich zum einen um die Maschinenprogrammierung, welche eine statische Ablauffolge generiert und zum anderen um die dynamischen Maschinenkommunikation, die es ermöglicht zu jeder Zeit Einfluss auf einen Fertigungsablauf zu nehmen.

2.1.1 MASCHINENPROGRAMMIERUNG

Die Maschinenprogrammierung stellt eine statische Form der Konfiguration von einer Fertigungsmaschine dar. Der Programmcode wird als PLC oder G-Code konformes Programm niedergeschrieben und in die Maschine eingespeist. Diese agiert nach den darin definierten Bedingungen. Die Maschinenprogrammierung besitzt starke Ähnlichkeit zu Interpreter basierten Plattformen, in welchen ein Skript sequentiell abgearbeitet wird. Im Folgenden werden die beiden Plattformen NC und PLC vorgestellt und ihre Programmierung in PLC oder G-Code. Diese sind durch ihren großen Marktanteil bedeutsame Faktoren, welche für die Entwicklung eines Datenhaltungskonzeptes berücksichtigt werden sollen.

2.1.1.1 Numerical Control (NC)

NC – Maschinen [4] [5] umfassen alle Werkzeugmaschinen, welche mittels eines numerischen Systems gesteuert werden. Es handelt sich hierbei typischerweise um achsbasierte Maschinen wie Mehrachsfräser oder Roboter. Die Aufgabe dieser Maschinen ist das Verfahren entlang einer berechneten Bahn, um mithilfe eines Werkzeuges einen erwünschten Effekt an einem Werkstück zu erzielen. Der erwünschte Effekt kann hierbei z.B. das Abtragen einer Oberfläche durch einen Bohrer sein oder auch das Biegen eines Stahlrohres mithilfe einer Bogenform als Werkzeug. Es wird, wie in *Abbildung 1* dargestellt, zuerst über eine menschliche Interaktionsschnittstelle ein Programm eingelesen, welches Befehle für das Verfahren der Maschine beinhaltet. Hierbei kann auf normierte Programmiersprachen wie die Step NC Normen der DIN 66025 zurückgegriffen werden, wie z.B. G-Code. Es ist ebenfalls möglich über einen „Teach in“ Prozess manuell Punkte oder ganze Strecken einzuspeisen. Die Maschine wird diese dann als Wiederholung abfahren. Ein solches „Teach in“ findet ausschließlich Verwendung wo kein hoher Grad an Präzision gefordert wird, zudem beschränkt sich diese Funktion auf die Nutzung von Roboter. Dies könnte z.B. das Umfahren eines Hindernisses sein, welches nicht in der Programmierung vorgesehen war. Diese codierten Befehle werden im Anschluss in den NC-Speicher geschrieben und danach in einen Satz-Puffer geladen. Ein einzelner Satz beinhaltet Punktinformationen, wie den nächsten Anfahrpunkt und die dazugehörigen Eigenschaften wie z.B. Geschwindigkeit. Der Puffer stellt die Quelle für das zyklisch betriebene Echtzeitsystem dar, welches für die Steuerung der Achsen zuständig ist. Diese Steuerung findet mit PLCs, Interpolationsmodulen,

Rückwärtstransformationsmodulen, Vorwärtstransformationsmodulen und Sensordatenverarbeitungsmodulen statt. Die Vorwärtstransformation bezeichnet hierbei die Umwandlung der Achsdaten in dreidimensionale X, Y und Z Werte in ein definiertes 3D System. Die Rückwärtstransformation wandelt im Gegenzug diese Werte in für die relativ zur Achse bestehendes Zahlensystem um. Da eine Kurve nicht exakt innerhalb eines deterministischen Echtzeitsystems berechnet werden kann, werden diese durch die Interpolationsmodule interpoliert. Um Schwingungen einer NC-Maschine auszugleichen oder den Verschleiß eines Werkzeuges zu kompensieren, werden Sensordaten innerhalb der Maschine gesammelt und an den Prozess als Feedback zurück geliefert.

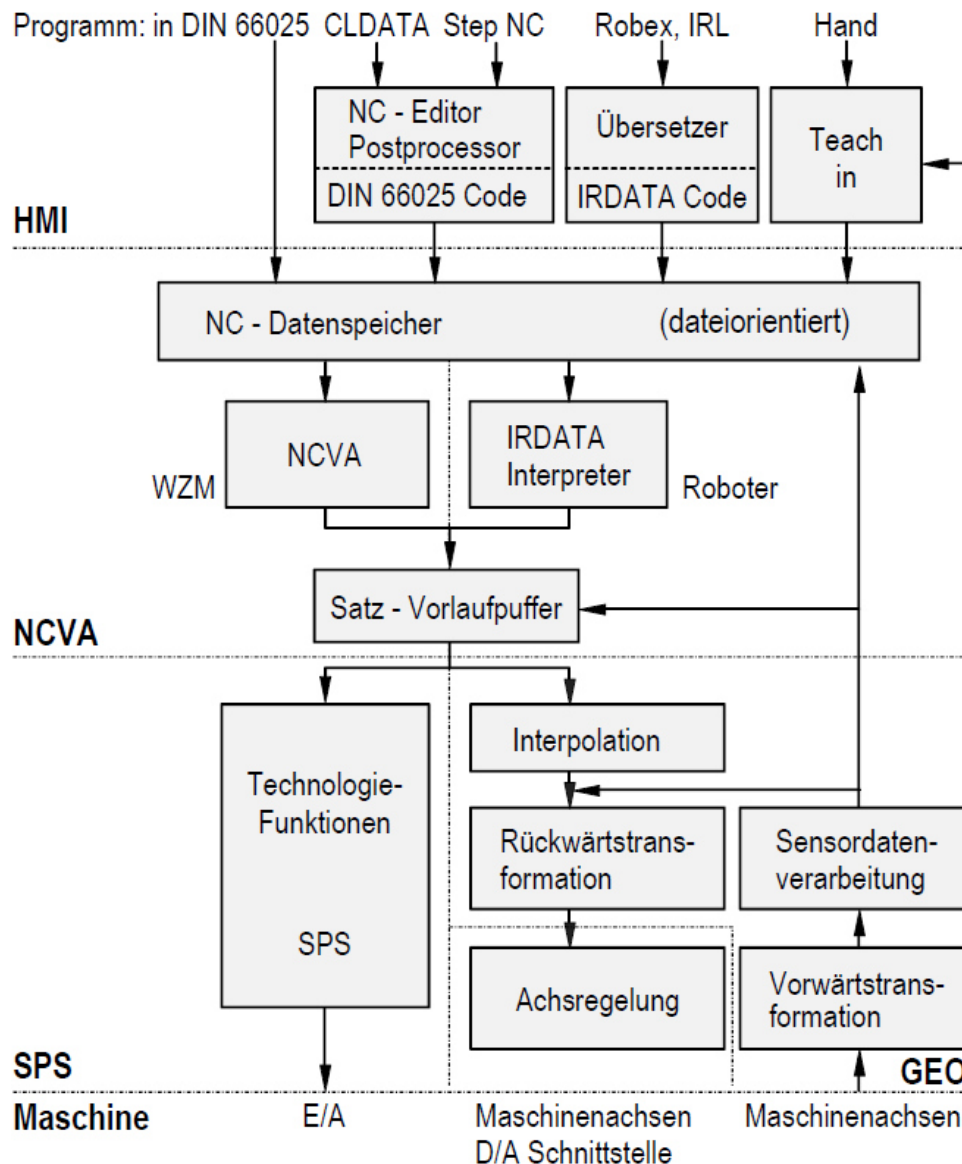


ABBILDUNG 1 NC ARCHITEKTUR [5]

2.1.1.2 Programmable Logic Controller (PLC)

PLC oder auch Speicher Programmierbare Steuerung (SPS) [5] [6] stellen ein Kernelement der heutigen Automatisierungsindustrie dar. Es handelt sich hierbei um Bauteile, welche einzelne Baugruppen oder modulare Steckverbindungen darstellen. Diese Bauteile besitzen Ein- und Ausgangsklemmen und stellen die Hardwarevertreter oder auch „Hard-PLC“ dar. Eine PLC kann zudem rein als Softwarelösung umgesetzt werden. Der Aufbau und Ablauf solcher als „Soft-PLC“ bezeichneten Systeme ist analog zu

einer „Hard-PLC“. Das „Soft-PLC“ System wird auf einem Echtzeitsystem betrieben und überträgt seine virtuellen Ein- und Ausgangsklemmen auf ein Bussystem, welches die Daten überträgt. Die Eingänge erhalten die Signale von Sensoren oder weiteren PLC-Geräten. Abhängig von den Belegungen der Eingangsklemmen, werden über das konfigurierte Steuerprogramm die Ausgangsklemmen mit Werten belegt. Hierdurch werden Steuerbefehlsempfänger, Aktuatoren oder weiter PLC Bausteine gesteuert. Die Steuerung erfolgt, wie in *Abbildung 2* dargestellt, über einen zyklischen Ablauf.

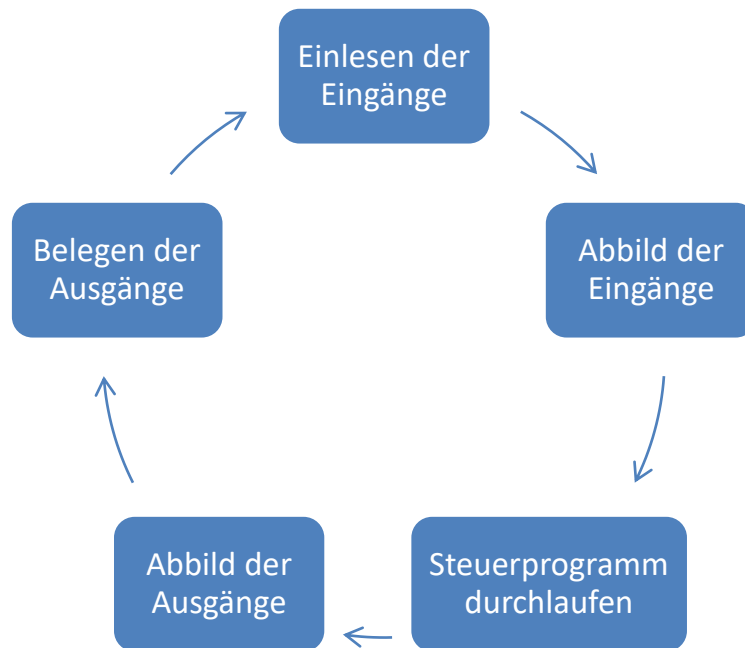


ABBILDUNG 2 PLC ZYKLUS

Hierbei wird initial jede Eingangsklemme mit ihrer Wertebelegung ausgelesen. Aus diesen Werten wird ein Abbild erstellt, welches den aktuellen Zustand der Eingänge für diesen einen Zyklus beschreibt. Im Anschluss wird ein PLC-Steuerprogramm ausgeführt, welches abhängig von dem Eingangsabbild Berechnungen durchführen kann und Belegungen für die Ausgangsklemmen in das Ausgangsabbild schreibt. Ein Zyklus wird über das Abschließen des PLC-Steuerprogramms und ein anschließendes Schreiben des Ausgangsabbildes auf die jeweiligen Ausgangsklemmen beendet. Jeder Zyklus unterliegt einem definierten Zeitfenster und hat somit immer dieselbe Zyklusdauer. Ein solches Zeitfenster liegt von unter einer Millisekunde bis 10 Millisekunden abhängig von der jeweiligen Konfiguration des Herstellers. Wird ein PLC-Steuerprogramm vorher beendet wartet die PLC mit der Ausgabe der Ausgangsbelegungen, bis die erwünschte Zykluszeit erreicht wurde. Ein PLC-Steuerprogramm, welches das gegebene Zeitfenster überschreitet, wird unterbrochen und mit seinem aktuellen Stand beendet. Dies ist zum einen eine potentielle Fehlerquelle im laufenden Betrieb, da so nicht vollständige Ausgangsbelegungen zustande kommen können, zum anderen aber auch eine notwendige Maßnahme um die deterministische Echtzeit des Systems zu wahren. Das Steuerprogramm einer PLC wird über eine separate Schnittstelle auf das Gerät geladen. Ein solches Programm sowie der Aufbau einer PLC wird in der Norm IEC 61131 definiert. Für die Programmierung stehen mehrere Standards zur Verfügung. Diese ermöglichen die graphische, wie in *Abbildung 3* links mit einem PLC Kontaktplan beispielhaft dargestellt, oder die rein textuelle Erstellung von PLC – Steuerprogrammen, abgebildet in *Abbildung 3* rechts mittels einer PLC Anweisungsliste. Trotz einer großen Anzahl an potentiellen Möglichkeiten eine PLC zu programmieren existieren vom Standard abweichende Umsetzungen, wie z.B. die an die Anweisungsliste angelehnte Sprache für Siemens S7. So ist ein PLC Standard konformes

Steuerprogramm, auf jedem PLC Gerät lauffähig aber nicht jedes PLC Steuerprogramm ist garantiert lauffähig auf einem anderen PLC Gerät, von welchem Ein- und Ausgangsklemmen sowie die Zykluszeit und Rechenleistung äquivalent sind.

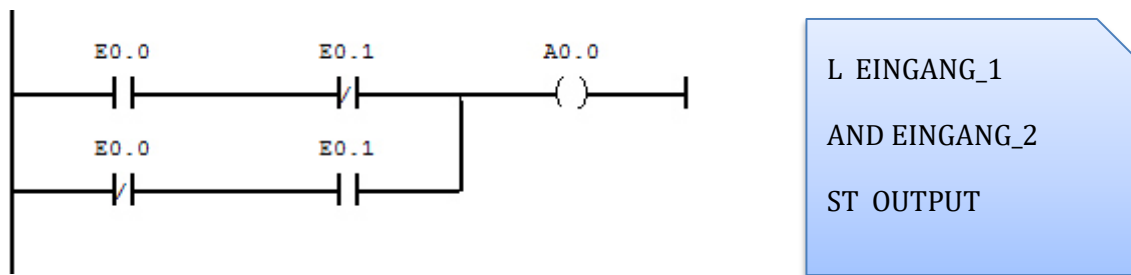


ABBILDUNG 3 KONTAKTPLAN [34] (LINKS) UND ANWEISUNGSLISTE (RECHTS)

2.1.1.3 G – Code

Dies ist ein in den Normen DIN-66025 sowie ISO-6983 definierter Standard für die Programmierung von NC-Maschinen. In *Abbildung 4* wird eine solche Liste an Anweisungen beispielhaft dargestellt. Diese wird sequentiell abgearbeitet und von der NC-Maschine ausgeführt. Der dargestellte Programmabschnitt beinhaltet, innerhalb der ersten Zeile N010, das Verfahren im Eilgang zu der Zielposition mit den Koordinaten X = 100 und Y = 100. Der Befehl für den Eilgang lautet hierbei G00. Die Anzahl der adressierbaren Dimensionen im Raum korreliert mit der Anzahl der operativen Dimensionen der NC-Maschine. Durch den Standard werden mehrere fundamentale Funktionsgruppen beschrieben, welche z.B. das Verfahren von Kurven oder Kreisen definieren lassen. Es folgt dabei dem Muster, nach welchem eine Funktionsgruppe solange mit einem Wert belegt bleibt bis dieser durch einen neuen Wert überschrieben wird. So wird in den Zeilen N020 mit dem Kommando Z0 das Verfahren zum Ursprung des Koordinatensystems gestartet. Da aber davor in der Zeile N010 der Eilgang durch G00 eingeschaltet wurde, verfährt die Maschine im Eilgang zum Ursprung. Durch diese Normen ist sichergestellt, dass jede NC-Maschine ein solches G-Code Programm einlesen kann und äquivalent umsetzt. Um jedoch die Fertigung weiter zu verbessern und eine optimierte Fehlerkorrektur zu erzielen, erweitern viele Hersteller von NC-Maschinen die Anzahl an möglichen Befehlen. Dies sind z.B. Befehle, um eine Wiederholung eines Befehlssatzes einzuleiten oder spezielle Verfahrrstrukturen in einem Kommando zu bündeln.

N010 G00 X100 Y100

N020 Z0

N030 G01 Z-2 F10

N040 G01 X110 F20

N050 Y200 F15

N060 G00 Z10

ABBILDUNG 4 G-CODE BEISPIEL

Dies wiederum hat zur Folge, dass zwar jedes G-Code Programm auf allen NC-Maschinen ausführbar ist jedoch nicht jedes NC-Programm auf G-Code Basis dasselbe Ergebnis bei jeder funktional äquivalenten NC-Maschine erzielt, da diese eventuell bestimmte Befehle nicht verarbeiten kann und diese somit ignoriert oder einen Fehler signalisiert.

2.1.2 MASCHINENKOMMUNIKATION

Die Maschinenkommunikation erlaubt im Gegensatz zur Maschinenprogrammierung die direkte Interaktion mit einer Maschine. Durch diese wird es möglich dynamisch zur Laufzeit Einfluss auf den Betrieb dieser zu nehmen und dessen Zustand auszulesen. Im Folgenden werden die Technologien OPC Classic, OPC UA und ADS vorgestellt, um einen Einblick in die Strukturen und Modelle hinter diesen zu erhalten.

2.1.2.1 Open Platform Communications (OPC Classic/UA)

Open Platform Communications (OPC) [7] findet seinen Ursprung in der Automatisierung und wurde zunächst „OLE for Process Control“ genannt. Es wird von der OPC Foundation spezifiziert und weiterentwickelt. Die Umbenennung resultierte aus der schwindenden Bedeutung von OLE-Objekten in diesem Bereich. OPC wird in zwei Versionen aufgeteilt, welche OPC Classic und OPC Unified Automation (OPC UA) darstellen. OPC Classic ist die erste Generation und diente dem Entwurf einer herstellerunabhängigen Echtzeitkommunikationsschnittstelle. Es wurden hierbei für verschiedene Anwendungsfälle getrennte Spezifikationen beschrieben, welche den Zugriff auf Datenelemente definieren. Drei der bedeutendsten Spezifikationen, innerhalb der Industrie, beschreiben den Zugriff auf Echtzeitdaten (OPC DA), die Handhabung von Events (OPC AE) und die Bereitstellung von Datenhistorien (OPC HDA). OPC Data Access dient dem Zugriff auf Variablen, welche von einem echtzeitfähigen Controller bereitgestellt werden. Dabei könnte es sich z.B. um eine SPS handeln. Die OPC DA Spezifikation nutzt hierfür eine lokale und an den OPC Client gebundene Gruppierung von Elementen, welche für den Client von Interesse sind. Der Zugriff erfolgt über die unterstützten Operationen Lesen, Schreiben und Abonnieren. Dies wird, wie in *Abbildung 5* illustriert, dadurch erreicht, dass der OPC DA Client erst den Namespace nach Elementen durchsucht und gezielt diese einer OPCGroup hinzufügt. Diese Gruppe besitzt eine Konfiguration, in welcher das zugewiesene Element für den OPC DA Client aktualisiert werden soll. Dabei werden nur Änderungen an den Client weitergeleitet. Um die Validität von Daten zu gewährleisten, werden diese Aktualisierungen mit einem Zeitstempel versehen. Hiermit sollen auftretende Fehler durch Verbindungsabbrüche umgangen werden.

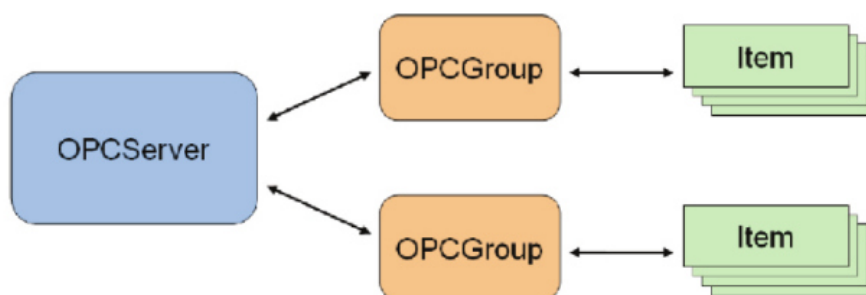


ABBILDUNG 5 OPC DATA ACCESS [7]

OPC Classic findet seine Hauptverwendung in der Anbindung von HMIs an eine echtzeitfähige Maschine, um innerhalb derer den aktuellen Status einer solchen Maschine darzustellen und Nutzereingaben in den Controller zu übermitteln. Der größte Vorteil von OPC Classic liegt in seiner

öffentlichen Spezifikation. Diese ermöglicht es jedem Gerätehersteller für sein jeweiliges Produkt einen OPC Treiber zu schreiben, der von jedem Gerätenutzer innerhalb eines OPC Classic Systems verwendet werden kann. Die Schwäche von OPC Classic liegt in seinem genutzten Kommunikationsprotokoll sowie seiner Client-Server Architektur. Es nutzt Microsofts DCOM als Kommunikationsprotokoll, welches keine plattformübergreifende Umsetzung von OPC Classic ermöglicht und nicht auf dem Ethernet-Standard aufbaut. Dies hat zur Folge, dass DCOM nicht über das Internet nutzbar ist und somit die Kommunikation auf ein lokales Netzwerk beschränkt. Zudem sind DCOM Systeme komplex zu konfigurieren und besitzen lange, nicht konfigurierbare Request Timeouts. Zudem resultiert aus der Client-Server Architektur eine statische Maschinenkonfiguration, welche stets bei Änderungen manuell neu konfiguriert werden muss und somit eine starke Kopplung erzeugt. Daher wurde von der OPC Foundation das neue OPC Unified Automation (OPC UA) Protokoll entwickelt und innerhalb der Standardreihe IEC 62541 standardisiert. Als Kommunikationsprotokoll nutzt OPC UA das TCP/IP – Protokoll anstelle von COM/DCOM und bietet somit die Fähigkeit jegliche Netzwerke zu nutzen, welche über Ethernet angebunden sind. Dies macht OPC UA zudem fähig Netzwerke über das Internet aufzuspannen. Im Gegenzug zu OPC Classic wird anstelle einer Client/Server-Architektur eine serviceorientierte Architektur (SOA) genutzt. Diese Umstellung ermöglicht es die starke Kopplung zwischen physikalischen Endpunkten in Netzwerken aufzulösen, in dem man diese mit einer logischen Adresse assoziiert und adressiert, welche über einen Service-Bus aufgelöst wird. OPC UA Systeme bieten binäre, hybride und auf Webservice aufbauende Kommunikationsschnittstellen an, welche TCP oder HTTP/HTTPS als Kommunikationsprotokoll nutzen. Es wird hierbei bei den Webservices auf eine XML-Repräsentation (UA XML), HTTP/HTTPS, SOAP und einer optionalen Sicherheitserweiterung WS-SecureCommunication gesetzt. Die binäre Schnittstelle stellt die auf Performance optimierte Kommunikationsschnittstelle dar. Diese nutzt TCP/IP als Basisprotokoll und erweitert es um eigene Headerfelder, dem UA TCP, um eine binäre Datenrepräsentation zu übertragen. Für eine sichere Kommunikation wird zudem das UA Secure Communication Protokoll verwendet, welches an WS-SecureCommunication angelehnt ist und denselben Zweck erfüllt. Die hybride Kommunikationsschnittstelle nutzt HTTPS für die Übertragung und Gewährleistung von Sicherheit. Das genutzte Datenformat ist dabei die binäre Datenrepräsentation. In *Abbildung 6* werden für eine verbesserte Übersicht alle drei der zur Verfügung gestellten Kommunikationsschnittstellen grafisch dargestellt. Die Modellierung von Daten erfolgt nicht mehr über separate Spezifikationen wie bei OPC Classic, sondern über ein einheitliches Metamodell, welches die Spezifikationen OPC DA, OPC AE, OPC Prozess und OPC HDA unterstützt. Wie in *Abbildung 7* visualisiert, werden diese über Knoten (Nodes) modelliert. Diese Knoten sind an Klassen aus objektorientierten Programmiersprachen angelehnt. Es wird stets von einer BaseNode abgeleitet, welche alle für die Adressierung notwendigen Attribute beinhaltet. Diese kann durch eigene Methoden, Variablen oder anderen Nodes erweitert werden. So bietet OPC UA über ein einheitliches Interface, den sogenannten Basis Services, alle Funktionalitäten von OPC Classic an und zudem eine flexible Erweiterbarkeit, durch eine Erweiterung dieser Basis Services.

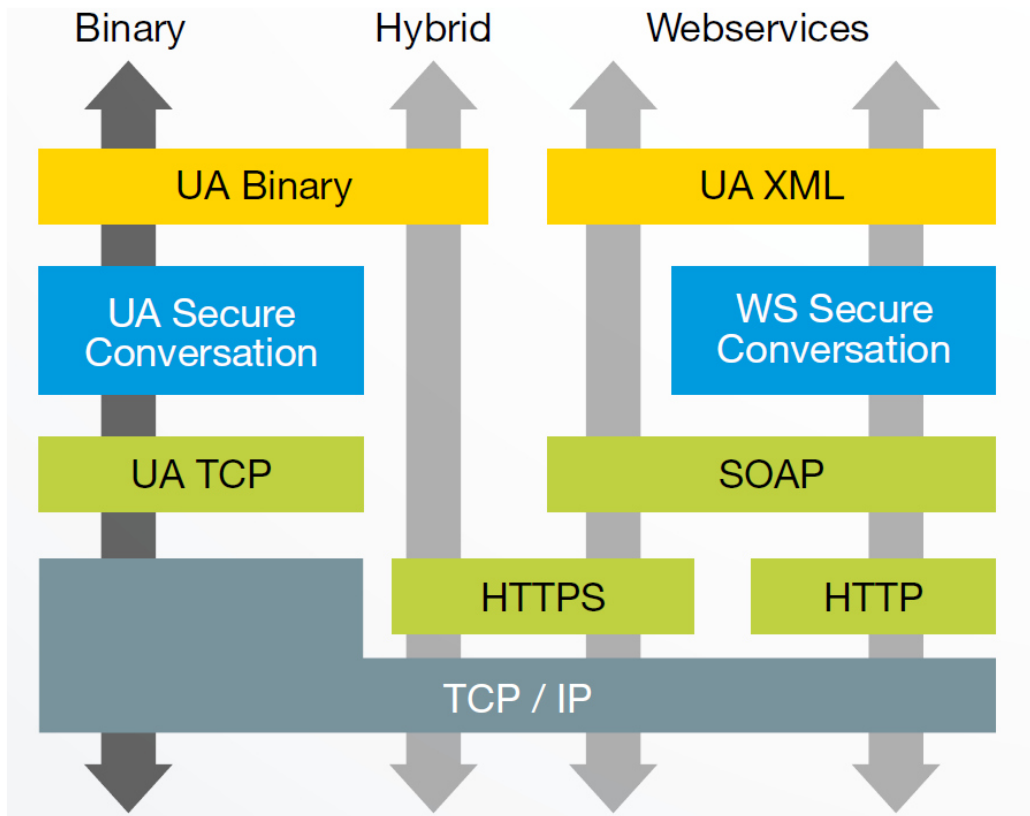


ABBILDUNG 6 OPC UA KOMMUNIKATION [8]

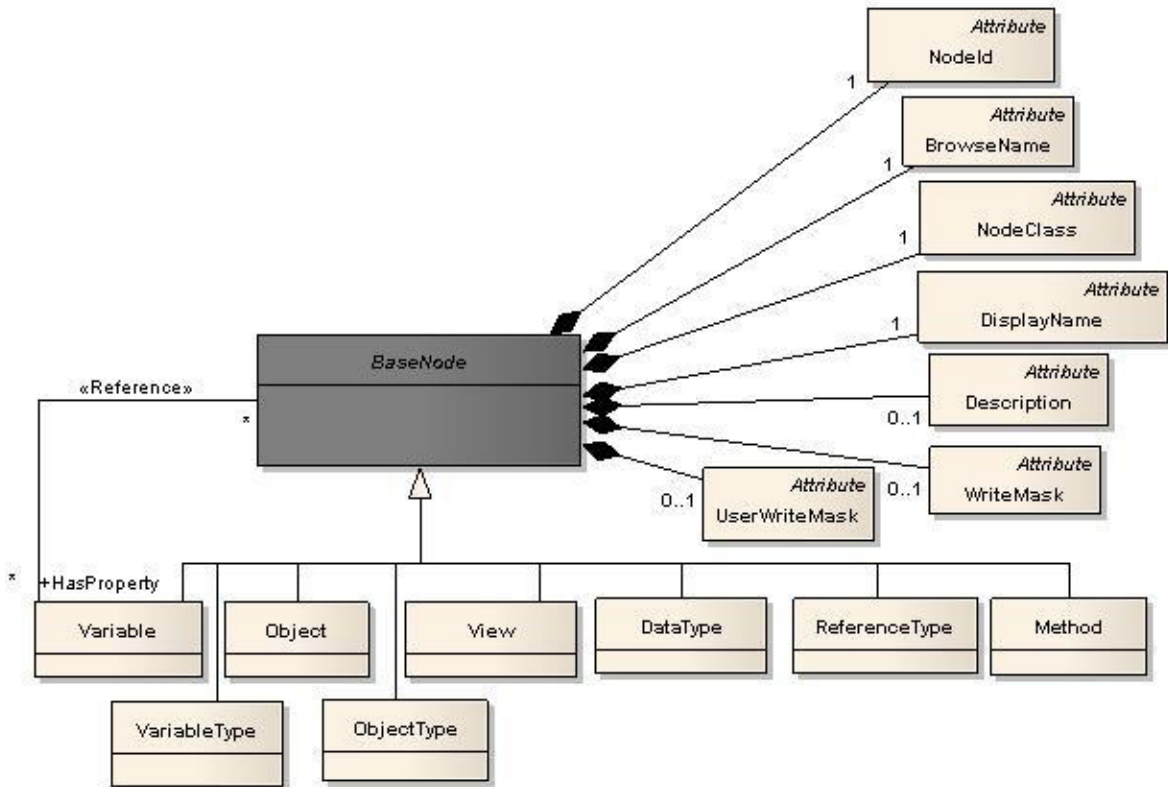


ABBILDUNG 7 OPC UA METAMODELL [9]

2.1.2.2 Automation Device Specification (ADS)

ADS ist eine von der Beckhoff Automation GmbH & Co. KG [10] [11] [12] entwickelte Kommunikationsschnittstelle. Sie dient dazu Echtzeitsysteme anzubinden. Der Aufbau besteht, wie in *Abbildung 8* dargestellt, aus einem zentralen Nachrichtenrouter, welcher alle Kommunikationsteilnehmer anbindet. Dieser sorgt für die Adressierung jedes einzelnen Endpunktes, welcher als Device bezeichnet wird. Die Interaktion mit Devices ähnelt dem Zugriff auf einen Shared Memory. Jedes ist Device auf einem physikalischen System über seinen eigenen ADS Port adressierbar, an welchen die Nachrichten verschickt werden. Es stellt ein verbindungsorientiertes Verfahren dar, an dem eine Update Cycle Time gekoppelt ist. Diese beschreibt die zyklische Aktualisierung der Abbildung innerhalb des Speichers. Einzelne physikalische Endpunkte werden über eigene Identifier adressiert, der sogenannten AMS Net ID. Durch die Kombination der AMS Net ID und eines ADS Ports können einzelne Devices angesprochen werden.

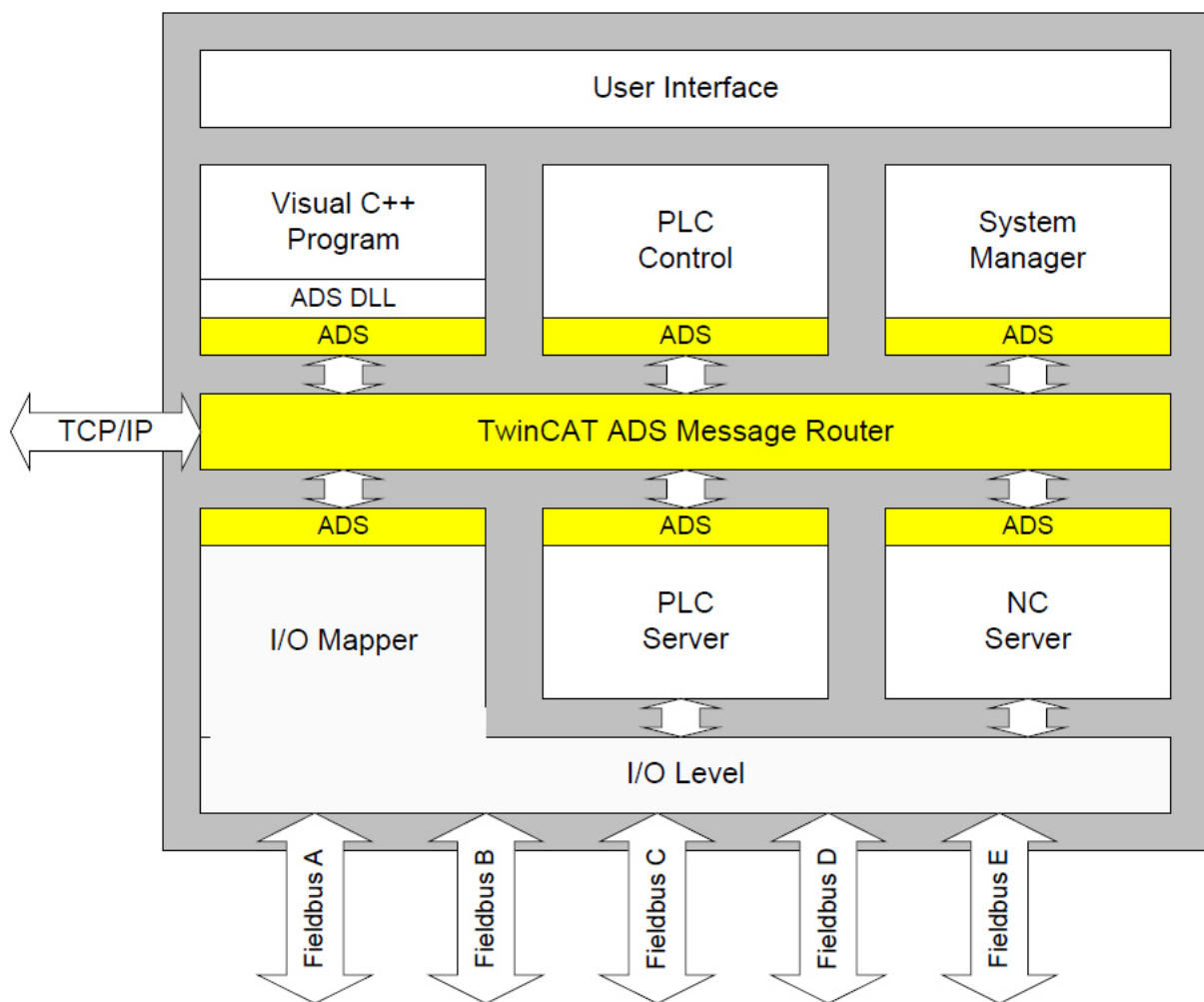


ABBILDUNG 8 ADS SYSTEMARCHITEKTUR [12]

Das ADS Interface beschränkt sich auf die beiden Interaktionen Lesen und Schreiben. Für eine bessere Übersicht werden Daten innerhalb von Gruppen gesammelt. Diese Gruppen besitzen einen Index und sollen eine semantische Kapselung innerhalb des Speichers darstellen. Innerhalb einer solchen Gruppe können anschließend Teile des Speichers ausgelesen oder überschrieben werden. Die Adressierung erfolgt hierbei über einen Index Offset, welcher beschreibt wie viele Speicherblöcke übersprungen werden sollen, bevor eine gewisse Anzahl an Blöcken ausgelesen oder überschrieben wird. Die Nutzung des Interfaces beinhaltet hiermit für die Adressierung des gewünschten Devices die AMS Net

ID, den ADS Port des Devices, einen Group Index, den Index Offset und eine Payload. Die Payload definiert beim Lesen die Länge der gewünschten Speicherblöcke und analog beim Schreiben die zu überschreibenden Speicherblöcke. Es existiert eine Richtlinie, welche eine Normierung der Portbelegungen beschreibt. So wird z.B. der ADS Port 110 für einen EventLogger reserviert oder alle Ports von 800-900 für die Anbindung von SPS Laufzeitsystemen [10]. Das genutzte Übertragungsprotokoll baut auf TCP/IP auf und überträgt TCP Ports auf ADS Ports. Es ist aber möglich ADS auf andere Ethernet basierende Feldbusprotokolle zu übertragen, hierdurch ist auch die Nutzung mehrere Protokolle gleichzeitig möglich, im Falle der genutzte ADS Message Router dieses unterstützt. Die Hauptverwendung findet es innerhalb des Beckhoff eigenen TwinCAT [11] Ökosystems und kann durch den Erwerb einer Lizenz auch von weiteren Industrieteilnehmern genutzt werden.

2.2 MODELLIERUNG

Dieses Kapitel sammelt aktuelle Modellieretechniken aus dem Softwaredesign und der industriellen Produktfertigung, um aufbauend auf diesen ein Modell zu entwickeln oder wiederzuverwerten, welches als Metametamodell für das angestrebte Datenhaltungskonzept genutzt werden kann.

2.2.1 UNIFIED MODELING LANGUAGE (UML)

UML [13] ist ein weitverbreitetes Werkzeug in der Systemmodellkonzeptionierung und der Visualisierung von Relationen zwischen Entitäten. Es ist ein Standardwerkzeug in der Softwareentwicklung und sollte Teil von jedem Spezifikationsdokument sein, welches Daten beinhaltet, die sich durch UML darstellen lassen. Dies resultiert aus den einheitlichen Definitionen durch den UML Standard, welcher zu einem einheitlichen Verständnis eines modellierten Systems beitragen kann und Mehrdeutigkeit von Spezifikationen verhindert. UML selbst folgt der Definition von Metamodellen und stellt eine Form der graphischen Modellierung dar. Die Eigenschaft eines Metamodells resultiert daraus, dass UML selbst, wie in *Abbildung 9* dargestellt, das Modell des Metamodells „Meta Object Facility“ (MOF) ist. So kann je nach Betrachtungswinkel ein Modell entweder ein strukturbeschreibendes Metamodell oder ein systembeschreibendes Modell darstellen. Im Falle von UML handelt es sich hierbei um eine Hierarchie, welche vier Ebenen besitzt. In der obersten Ebene befindet sich das MOF [14] als Metametamodell, gefolgt von UML als Metamodell mit welchem im Anschluss das Benutzermodell als Modell erstellt werden kann. Die letzte Ebene stellt die Instanzebene dar, welche für die abstrakt definierten Elemente eines Modells Laufzeitobjekte instanziiert. All diese vier Ebenen folgen demselben Prinzip, dass jedes Element innerhalb dieser Ebene eine Instanz von einem Element aus der darüber liegenden Ebene darstellt. Dies wiederum bedeutet, dass es somit auch immer eine semantische Bedeutung besitzt, welche in dieser überliegenden Ebene definiert wurde. Zu einem Metametamodell, Metamodell und einem Modell gehören zudem Dokumente. Diese Dokumente spezifizieren die semantische Bedeutung von Elementen sowie die Bedeutung der Relationen zwischen diesen Elementen innerhalb der Modelle. Ein solches Dokument könnte z.B. die Beschreibung einer Klasse mit optionalen Attributen enthalten, welche nur unter definierten Umständen während der Instanziierung auftreten. UML bietet innerhalb seines Standards mehrere verschiedene Metamodelle für verschiedene semantische Aussagen. Jedes beschränkt sich auf die Spezifizierung bestimmter Teilbereiche der Softwareentwicklung. So bieten Entity-Relationship-Diagramme die Möglichkeit die Datenstrukturen von relationalen Datenbanken wie SQL zu modellieren.

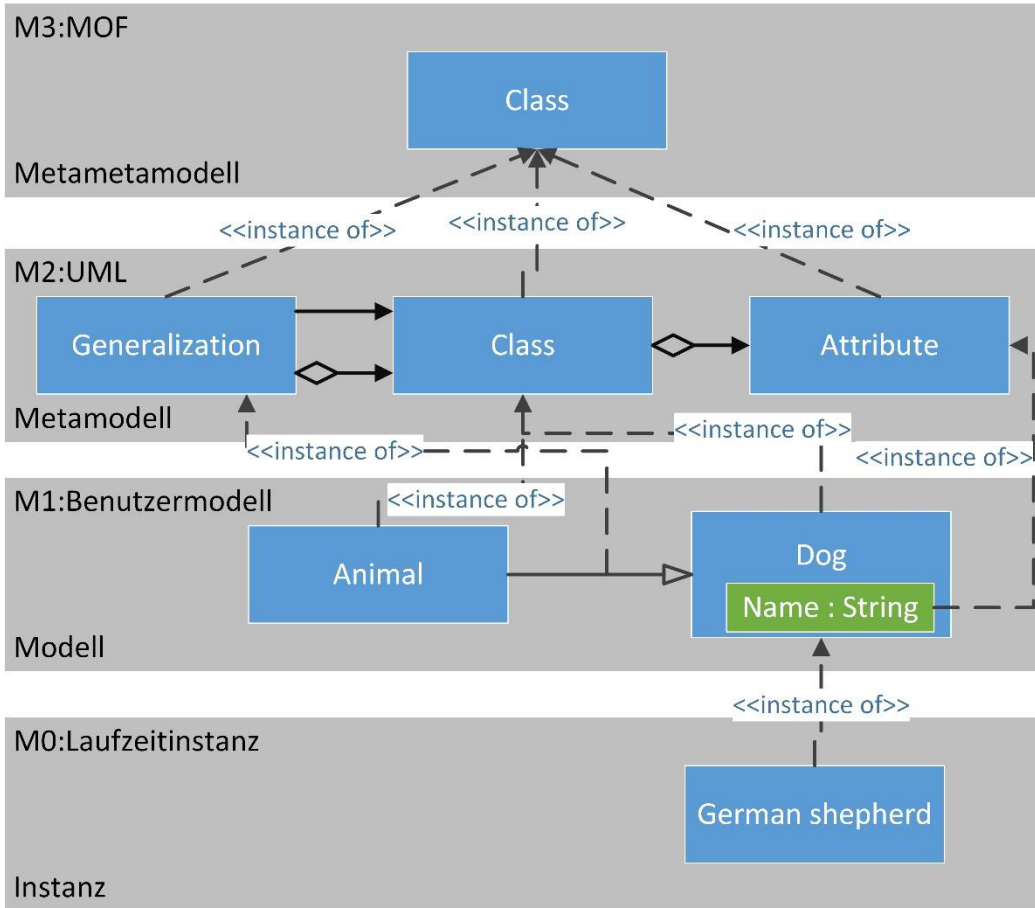


ABBILDUNG 9 UML METAMODELLHIERARCHIE

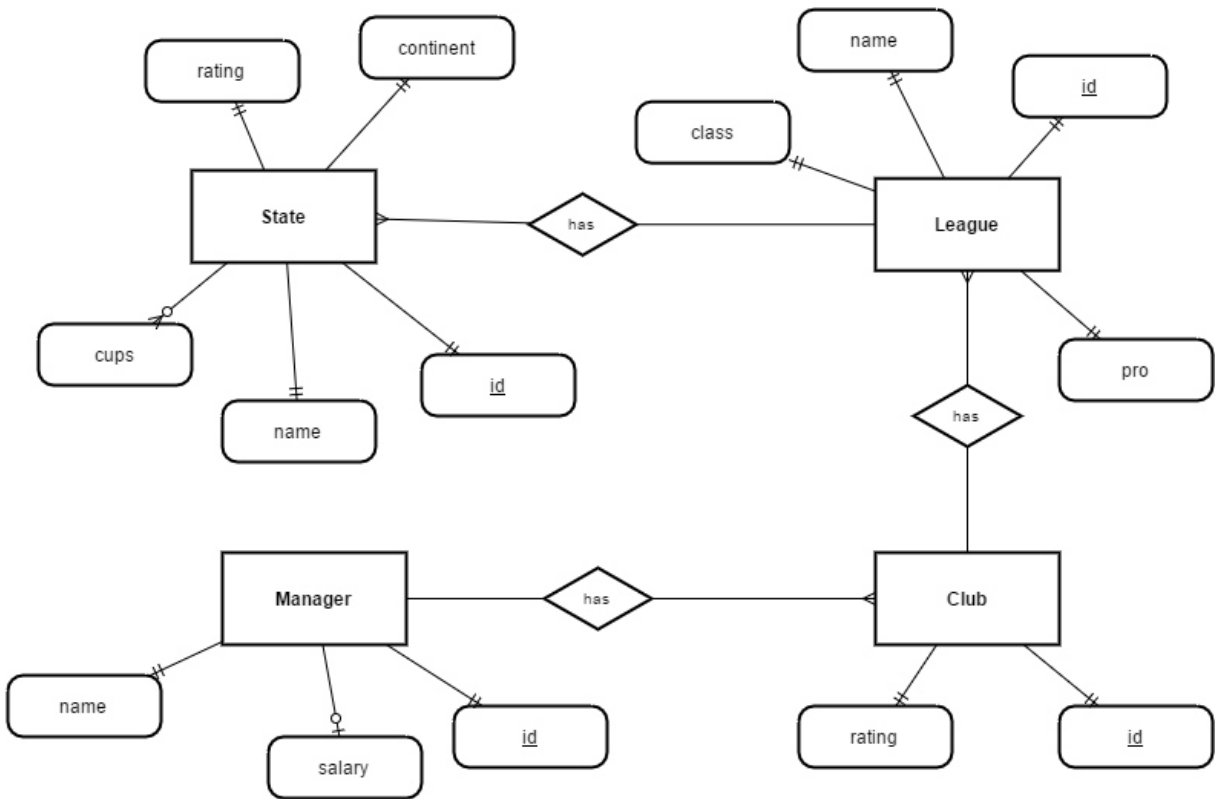


ABBILDUNG 10 ENTITY-RELATIONSHIP

Als Beispiel wird in *Abbildung 10* die Datenbank eines Clubmanagementsystems visualisiert. Die rechteckigen Objekte stellen die Entitäten dar und die elliptischen die jeweiligen Attribute. Unterstrichene Attribute stehen für den Primärschlüssel eines Tabelleneintrages. Die Entitäten sind über die Relation „has“ miteinander verbunden. Im Kontrast zu ER Diagrammen ist es auch möglich in UML nicht nur Relationen darzustellen, sondern auch komplexe Klassen Strukturen oder den Datenfluss eines Programmes. In *Abbildung 11* wird der Ablauf eines simplen „Hello World“ – Programmes dargestellt, welches den Ausgabe Text 100-mal versucht auszugeben, wenn die Zählvariable „Counter“ ungerade ist. Es stellen hierbei Ellipsen mit schmalen Linien den Startpunkt eines Diagramms dar, Rechtecke sind Aktionen welche ausgeführt werden, Rauten bilden Entscheidungspunkte ab und Rechtecke mit stark abgerundeten Ecken die Endpunkte eines Diagramms. UML Diagramme besitzen eine Vielzahl an Möglichkeiten einen sehr spezifischen Betrachtungswinkel von einer Datenmenge zu visualisieren und somit eindeutig zu beschreiben. Weitere mögliche Diagrammformen stellen z.B. Sequenzdiagramme, Komponentendiagramme oder Use-Case-Diagramme dar, welche durch UML beschrieben werden können.

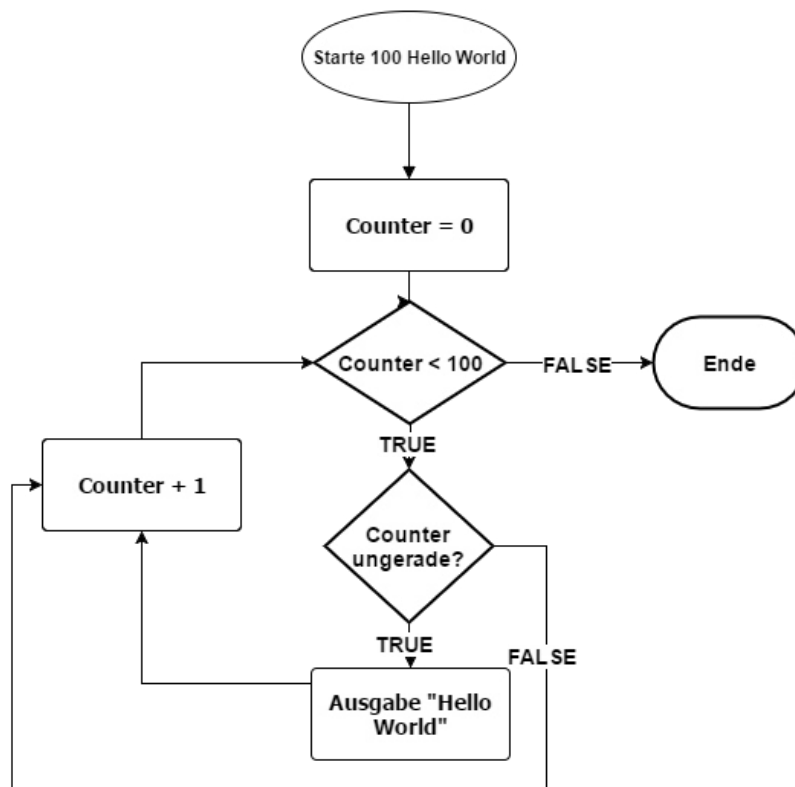


ABBILDUNG 11 FLUSSDIAGRAMM IN UML

2.2.2 BUSINESS PROCESS MODEL AND NOTATION (BPMN)

BPMN [15] ist eine graphische Modellersprache, deren Spezialisierung auf der Modellierung von Geschäftsprozessen liegt. Es besitzt die größte Ähnlichkeit zu UML-Flussdiagrammen, da es hier auch um die Modellierung von Aktivitäten und Entscheidungen geht, welche auf dem Abfluss zwischen dem Start und dem Ende eines Prozesses stattfinden. Dabei ist BPMN reichhaltiger an möglichen Modelliermöglichkeiten, welche im Anschluss erläutert werden. Diese Modellersprache wird wie UML von der Object Management Group verwaltet und definiert. So ermöglicht es verschiedene Domänen eines Prozesses, Unterprozesse, komplexe Bedingungen und Parallelität abzubilden. BPMN befindet

sich zum Zeitpunkt dieser Arbeit in der Version 2.0. Im Folgenden werden die hierfür bereitgestellten Überkategorien an Symbolen präsentiert und an einem Beispiel für ihre Verwendung illustriert.

Pools und **Lanes** dienen der Strukturierung von Prozessumgebungen. Ein abgebildeter Geschäftsprozess bildet die Interaktion zwischen den beteiligten Teilnehmern innerhalb des Geschäftsprozess ab. Ein solcher Teilnehmer, meist ein Unternehmen oder eine Abteilung eines Unternehmens, wird über einen Pool dargestellt. Dies dient der Kapselung von Zuständigkeit. Ein Pool kann mehrere Lanes besitzen, welche sich in der Länge über die des umschließenden Pools erstreckt. Eine Lane dient der tieferen Strukturierung eines Pools, um die Zuständigkeit oder den Kontext einer Aktion oder Prozesses zu spezifizieren. *Abbildung 12* bildet die zugehörigen Symbole ab.

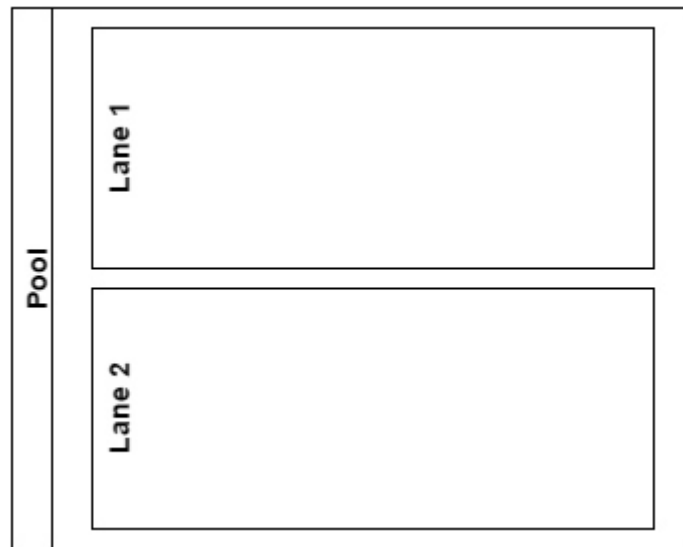


ABBILDUNG 12 BPMN POOLS UND LANES

Aktivitäten sind alle möglichen Formen von Subprozessen, welche für die Ausführung des gesamten Prozesses nötig sind. Es ist möglich einzelne atomare Aufgaben über Tasks abzubilden sowie weitere semantische Prozesseigenschaften darzustellen. Hierbei könnte es sich z.B. um Transaktionen, Prozesse mit menschlichen Akteuren, zeitgesteuerte Prozesse, nachrichtengesteuerte Prozesse oder Prozesse mit weiteren Unterprozessen handeln. All diese Aktivitäten werden, wie in *Abbildung 13* mit ihren Symbolen illustriert und über Rechtecke mit abgerundeten Ecken dargestellt. Für die jeweilige semantische Deutung besitzen diese verschieden gezeichnete Konturen und zusätzliche Symbole.

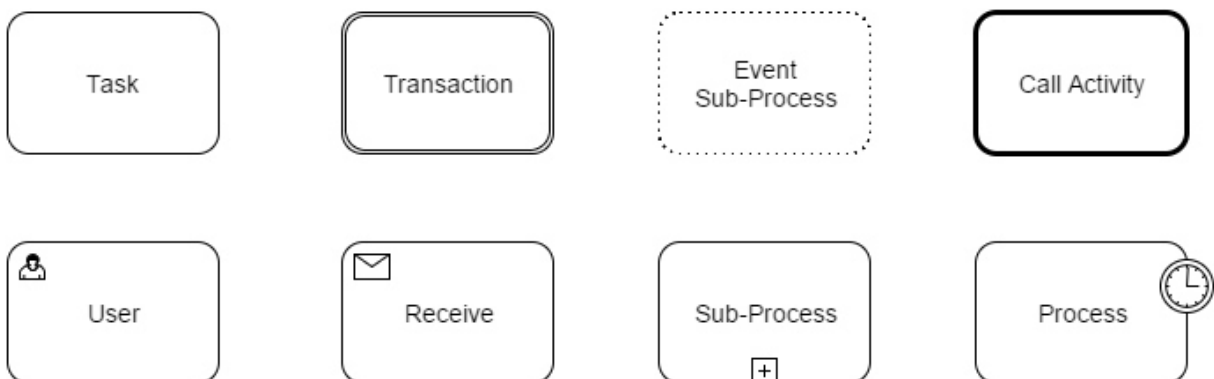


ABBILDUNG 13 BPMN ACTIVITY

So werden z.B. Aktivitäten, welche weitere Prozesse starten (Call Activity) mit einer starken Umrandung illustriert, da diese eine vergleichbare Aufgabe besitzen wie Events, welche einen Prozess starten. Würde eine solche Aktivität auch einen abgebildeten Unterprozess besitzen, so würde sie zusätzlich mit dem „+“ – Symbol für Subprozesse ausgedeutet werden. Aktivitäten werden über Flows miteinander verbunden und bilden hierdurch in Kombination mit Events einen Prozess.

Flows dienen der Orchestrierung von Aktivitäten und Events. Es ist möglich simple sequentielle Abläufe eines Prozesses über die Nutzung von Sequenz Flows zu modellieren. Existiert nach einer Entscheidung eine Ausführung, welche als natürliche Ausführung bezeichnet werden kann, wird diese über ein Default Flow signalisiert. Dies ist weitestgehend optional, erhöht aber die Lesbarkeit von BPMN – Diagrammen, da ersichtlich ist was der erwünschte Ablauf eines Prozesses ist. Es ist zusätzlich möglich einzelne Bedingungen, welche an Aktivitätsübergängen erfüllt sein müssen, über Condition Flows zu repräsentieren. Aktivitäten deren Ablauf nicht zeitlich über eine Sequenz erfolgt, sondern über das Eintreffen von Nachrichten deren zeitliches Eintreffen durch die „Loose Coupling“-Eigenschaften nicht eindeutig bestimmbar sind, können über Message Flows abgebildet werden. Alle Flows werden über monodirektionale Pfeile dargestellt. *Abbildung 14* illustriert die vier verfügbaren Flow-Arten mit ihren zugehörigen Symbolen.



ABBILDUNG 14 BPMN FLOW

Events, welche ausschnitthaft in *Abbildung 15* abgebildet sind, stellen gekapselte und lokalisierbare Ereignisse dar. Diese können das Resultat einer Aktivität darstellen oder von externen Prozessen ausgelöst werden. Sie lassen sich in drei Kategorien einteilen: Startevents, finale Events und Zwischenevents. Start Events sind Ereignisse, welche den Start eines Prozesses darstellen. Als Beispiel kann der abstrakte Beginn eines Subprozesses oder das Erhalten einer Nachricht dienen. Sie werden stets über Kreise mit dünnen Konturen gezeichnet. Wird das Ende eines Prozesses dargestellt, sind diese Konturen des finalen Events stärker betont. Es kann sich hierbei um ein abstraktes Ende eines Subprozesses handeln oder das Terminieren eines Prozesses, welcher abschließend eine Nachricht versendet. Die letzte Kategorie sind die Zwischenevents. Diese signalisieren entweder rein abstrakt,

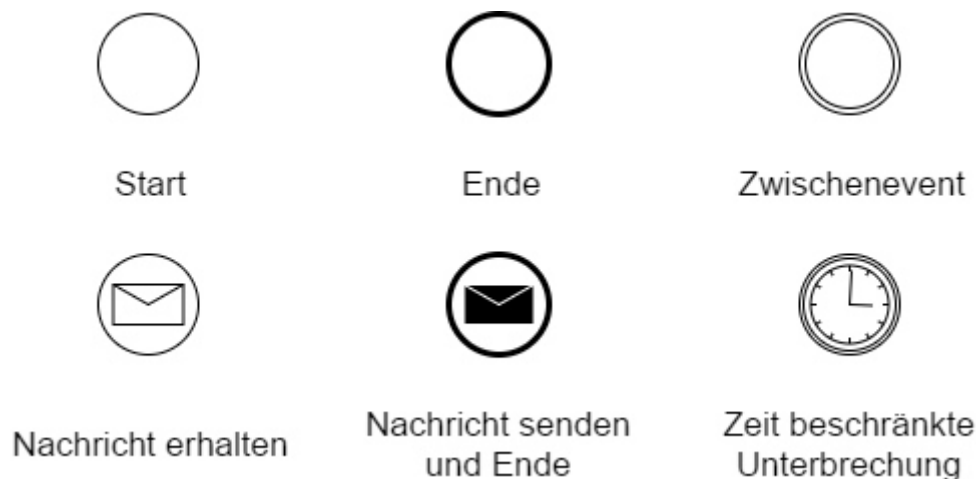


ABBILDUNG 15 BPMN EVENTS

dass es einen semantischen Punkt mit einer spezifizierten Bedeutung innerhalb eines Prozesses gibt oder sie sind wie das „Zeitbeschränkte Unterbrechung“ – Zwischenevent mit einer definierten Bedeutung versehen. Hierbei handelt es sich um ein Zwischenevent, welches über einen Zeitraum den Ablauf an seiner ausgelösten Position pausiert. Zwischenevents besitzen dünn gezeichnete Konturen aus zwei Linien. Die hier erwähnten Events sind bis auf ihre Kategorisierung nicht vollständig, da für jedes Ereignis ein eigenes Event erstellt werden kann. BPMN in der Version 2.0 umfasst mehr als 60 verschiedene Events mit einer definierten Bedeutung.

Gateways dienen der Darstellung von Entscheidungen, welche den Ablauf eines Prozesses beeinflussen. Sie werden über Rauten dargestellt und erhalten ihre Bedeutung rein aus einem zugehörigen Symbol in der Mitte. Die innerhalb der *Abbildung 16* dargestellten Gateways repräsentieren einen Ausschnitt der zur Verfügung stehenden Elemente. So wird eine binäre Entscheidung über ein leeres oder XOR Gateway dargestellt. Das AND – Gateway steht für die parallele Ausführung mehrere Flows und kann auch zur Synchronisation von mehreren Flows genutzt werden. Das OR – Gateway dient der Darstellung von definierten Entscheidungspfaden, welche dem Durchlaufen eines Switch-Case in der Programmierung ähnelt. Dies wird ergänzt über das Complex – Gateway, welches komplexe Entscheidungen symbolisiert. Diese können auch darin resultieren, dass multiple Flows parallel gestartet werden. Abschließend kann über das Messageeventbased – Gateway eine Entscheidung an den Empfang von Nachrichten gekoppelt werde, um daran korrelierende Entscheidungen zu repräsentieren.

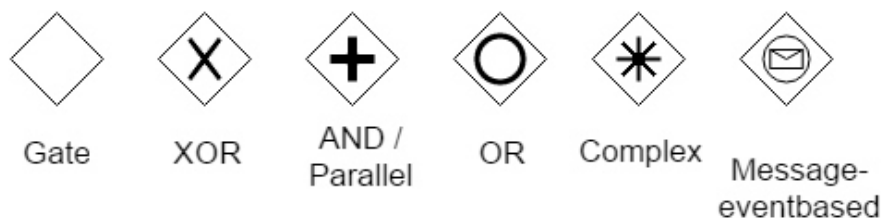


ABBILDUNG 16 BPMN GATEWAY

2.2.3 COMPUTER - AIDED X (CAX)

In der heutigen industriellen Fertigung sind Computer nicht nur in der Berechnung und Ausführung von definierten Prozessen anzufinden. Sie sind auch ein wichtiger Bestandteil der Produkt- und Anlagenentwicklung und somit der Erstellung solcher auszuführenden Aufgaben. Diese Form der computergestützten Entwicklung zieht sich durch alle industriellen Bereiche der Fertigung. Sie beschreiben jedoch alle das gleiche Prinzip, denn jeder CAX-Bereich beschreibt kein Standardformat, sondern dient nur als Oberbegriff für eine Gruppe von Softwarewerkzeugen und ihrer digitalen Speicherformate. Die Klassifizierung eines solchen Werkzeuges erfolgt somit über seinen Nutzen und den Bereich in welchem dieser sich auswirkt. So ist es auch möglich, dass ein Werkzeug zu mehreren Bereichen gehören kann. Im Folgenden werden die beiden CAX-Kategorien Computer-aided design und Computer-aided manufacturing näher betrachtet. Diese decken dabei nicht das komplette CAX Spektrum ab und dienen als Beispielvertreter. Weitere Klassifizierungen wären unter anderem Computer-aided Engineering (CAE), Computer-aided Innovation, Computer-aided Planning (CAP), Computer-aided Process Planning (CAPP), Computer-aided Quality Assurance (CAQ) und Computer-aided Styling (CAS).

2.2.3.1 Computer-aided design (CAD)

Das Ziel von CAD-Anwendungen [16] [17] ist die Erstellung von zwei- oder dreidimensionalen Objekten. Diese werden meist aus simpleren Objektstrukturen zusammengesetzt und mit der Hilfe von Manipulationsfunktionen verändert. Diese umfassen z.B. das Abrunden von Ecken, um hierdurch scharfe Kanten eines Kubus zu entfernen. Es ist dabei nicht unüblich, dass sich dieser Bereich nicht nur auf die Modellierung eines dreidimensionalen Objektes in seiner Form beschränkt. Eine Teilmenge der Eigenschaften des angestrebten Objektes können ebenfalls definiert werden. Es können z.B. mehrere Objektentitäten zu einer zusammengehörigen Entität zusammengefasst und deren Fertigungsmaterial als Eigenschaft angefügt werden. So sind CAD-Daten oft die Basis für Simulationen für die Eigenschaften eines modellierten Objektes unter dem Gesichtspunkt angestrebter Zielparameter. Die hier existierenden Datenformate sind vielfältig und umfassen die simple Konstruktion von zweidimensionalen Skizzen bis hin zu dreidimensionalen Objekten, welche aus mehreren mit Relationen zueinander verschachtelten Teilobjekten bestehen. *Abbildung 17* visualisiert ein mithilfe von dem CAD-Softwarewerkzeug CATIA [18] erstellte Fertigungsstraße. Hierbei erhalten verschiedene Materialien eine eigene Farbkodierung. Abgebildet sind in blau Fertigungsbänke, welche über eine Fertigungsstraße, mithilfe eines Fließbandes, von einer Fertigungsmaschine zur nächsten transportiert werden. Das Ziel von solchen CAD Resultaten ist nicht immer dasselbe. Es werden CAD Konstruktionen als Diskussionsbasis, Simulationen oder als Ausgangspunkt für CAM – Prozesse erstellt. Das angestrebte Ziel korreliert stark mit der aktuellen Position innerhalb der Fertigungskette.

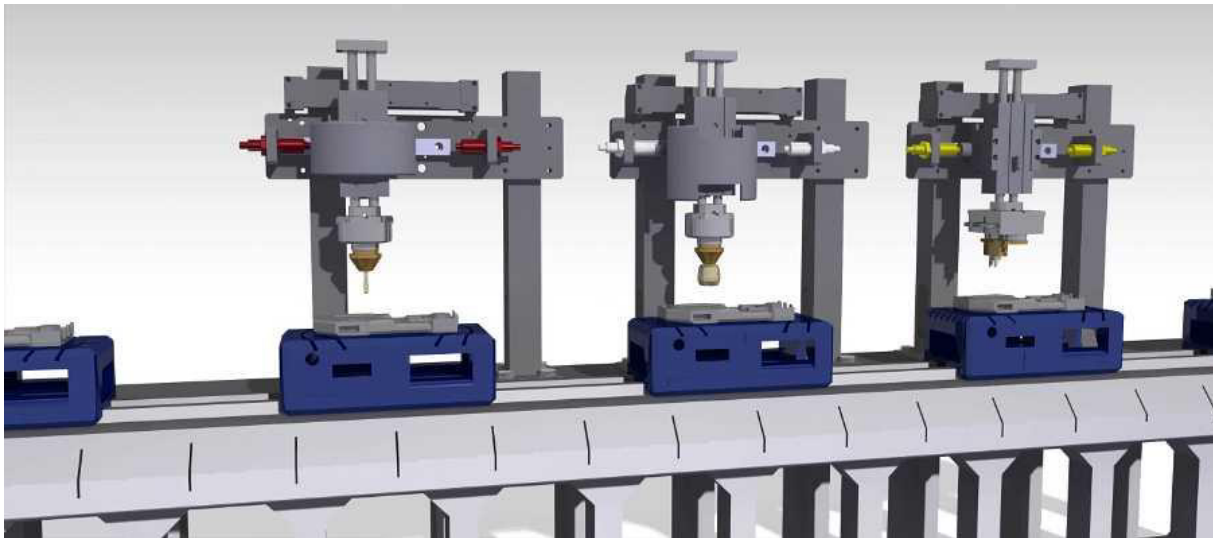


ABBILDUNG 17 CAD CATIA BEISPIEL [18]

2.2.3.2 Computer-aided manufacturing (CAM)

Die computergestützte Fertigung [19] bedient sich der in Kapitel 2.2.3.1 beschriebenen CAD-Daten und generiert aus diesen die Fertigung des darin konstruierten Objektes oder Teile daraus. Für eine solche Generierung werden ein Modell der fertigenden Akteure, ein Modell über die genutzten Ausgangsmaterialien und das angestrebte CAD-Objekt benötigt. Die fertigenden Akteure können dabei Werkzeugmaschinen wie NC oder Roboter darstellen. Ein Modell dieser Akteure hängt von der jeweils genutzten CAM-Software ab. Da hier ähnlich zu CAD kein Standardformat existiert, findet eine Vielzahl von CAM-Software und ihrer proprietären Dateiformate Verwendung in der Industrie. Sie beinhalten meist eine räumliche Darstellung der Akteure, unter Umständen auch als separates CAD-Objekt konstruiert, die Eigenschaften der von diesen genutzten Werkzeugen und die räumlichen Freiheitsgrade der Akteure. Das Modell für die Ausgangsmaterialien beschreibt lediglich die Beschaffenheit dieser und deren Materialeigenschaften. Es kann sich hierbei wiederum um ein CAD-

Objekt handeln. Auf Basis dieser zwei Modelle wird ein Fertigungsverfahren generiert, welches als Resultat das angestrebte Ziel-CAD-Objekt erzeugt. Dieser Vorgang kann vollständig automatisiert erfolgen oder auch menschliche Unterstützung benötigen. Dies hängt von der genutzten CAM-Software und der Komplexität der fertigenden Akteure ab. Der Vorteil von der Nutzung einer solchen CAM-Software liegt in der Möglichkeit der Simulation einer Fertigung. So kann die Fertigung auf einer Werkzeugmaschine simuliert werden und benötigt so weder kostenintensive Ressourcen noch den realen Besitz dieser Maschine. Es werden zusätzlich, durch die Berechnung potentieller Kollisionen, Beschädigungen an der Werkzeugmaschine und potentielle Fehlerquellen durch menschliche Interaktion vermieden. Das Resultat ist meist ein direkt auf der Maschine ausführbares Programm, wie der in Kapitel 2.1.1.3 vorgestellte G – Code für NC - Maschinen. *Abbildung 18* zeigt einen Ausschnitt der CAM – Software NX von Siemens, welche das Abtragen auf einem Bogen simuliert. Auf der rechten Seite der Abbildung kann man den hierfür generierten G – Code einsehen und diesen manuell anpassen, jegliche Änderungen werden in die Simulation übernommen. Dies ermöglicht es Kollisionen oder Abweichungen durch eine farbliche Hervorhebung zu erkennen.

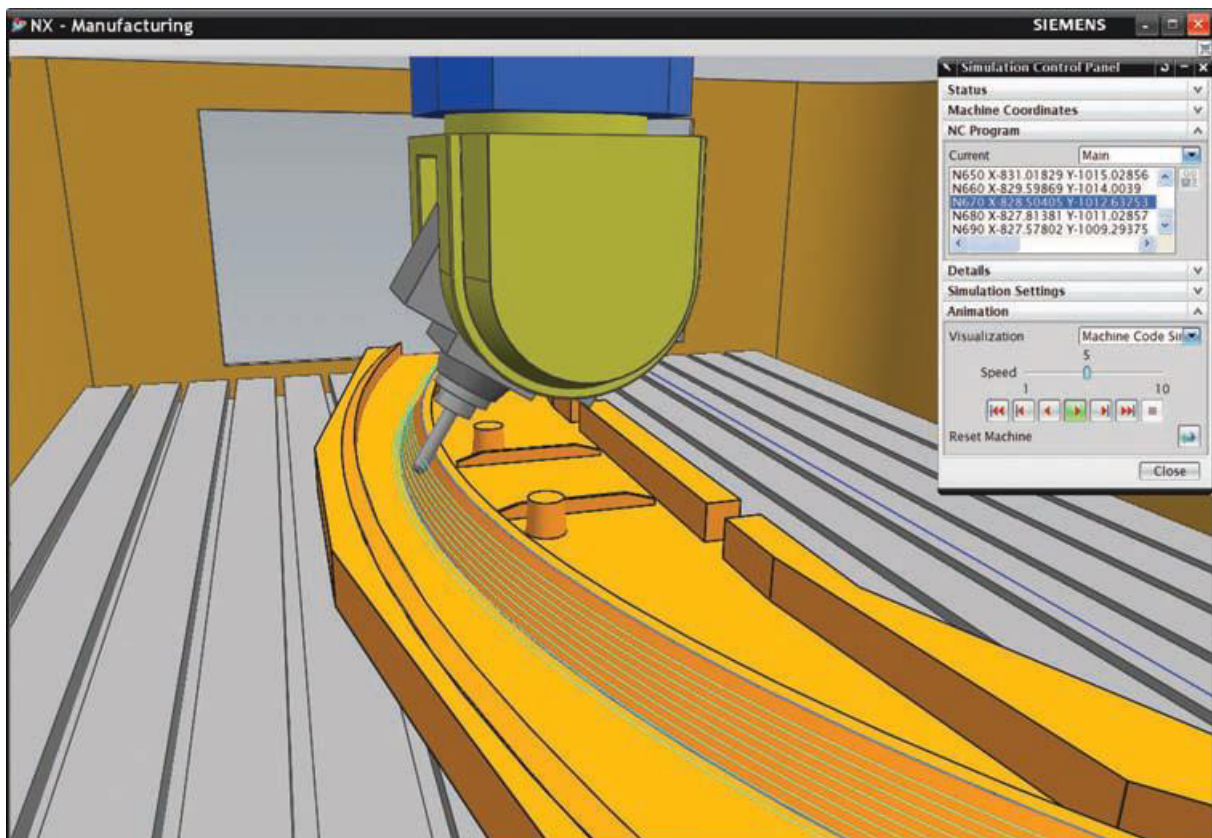


ABBILDUNG 18 NX SIEMENS [33]

2.2.4 AUTOMATION MOCKUP LANGUAGE (AUTOMATIONML)

AutomationML [20] [21] [22] [23] [24] ist eine auf XML basierende textuelle Modellierungssprache, welche einen objektorientierten Ansatz besitzt. Es baut selbst auf dem hierarchischen Modell CAEX auf. Das Ziel von AutomationML ist es verschiedene Bereiche der industriellen Fertigung zu verbinden, indem es ermöglicht ein einheitliches Datenformat bereit zu stellen. Hierbei soll AutomationML das Datenformat sein, mit welchem die Daten von einer Instanz, wie z.B. der Produktplanung in die Anlagenplanung überführt werden kann. AutomationML setzt hierbei auf eine Modellierung in 2 Phasen. Innerhalb der Deklarationsphase werden die Objekte mit ihren Strukturen beschrieben, dies

ähnelt der Java Nutzung von Klassen und Objekten, im Anschluss wird mit Hilfe dieser definierten Strukturen Objekte in einer Instanz-Hierarchie beschrieben. Die Deklarationsphase besteht aus drei Komponenten der Interfaceklassenbibliothek, der Rollenklassenbibliothek und der System-Unit-Klassenbibliothek.

Interfaceklassenbibliothek: Diese dient der Sammlung von Schnittstellen, welche ein System als typisierte Schnittstelle anbietet. Ein Beispiel wäre eine physikalische Schnittstelle für die Ein- oder Ausgangsklemmen einer PLC. Eine logische Schnittstelle könnte z.B. die Verbindung zweier Prozessschritte sein, wobei der durchlaufene Prozess eine „Ende“-Schnittstelle und der zu startende Prozess eine „Start“-Schnittstelle besitzen könnte. Die Aufgabe dieser Bibliothek ist das Definieren generalisierter Schnittstellen, welche durch das Prinzip der Vererbung Attribute und semantische Informationen weitergeben, um somit Generalisierung und Wiederverwertbarkeit zu erreichen. Hierbei definierte Interfaces können Attribute und zusätzliche Interfaces besitzen, welche somit eine „Is Part of“ Relation erhalten. Diese „Is Part of“ Relation dient der Strukturierung von Interfaces. Eine Interfaceklassenbibliothek besitzt immer eine Version, um Änderungen darin zu ermöglichen und diese über eine höhere Versionsnummer zu repräsentieren. Als Beispiel wird in *Abbildung 19* eine Interfaceklassenbibliothek mit dem Namen „InterfaceClassLib“ visualisiert. Sie besitzt die Version 1.0.0, welche die initiale Versionsnummer darstellt und zusätzlich das Interface „ProzessStartInterface“. Dieses Interface beinhaltet das Attribut „StartAttribut“. Das Attribut „StartAttribut“ ist vom Typ XML-Zeitstempel und dessen Maßeinheit erhält die Bezeichnung „ZEIT“. Der „Default Value“ für dieses Attribut ist „2016-08-20T15:30:44.2428373+02:00“. Ein „Default Value“ ist eine spezielle AutomationML Entität für Attribute, um deren initiale Wertebelegung zu definieren.

```

35     <InterfaceClassLib Name="InterfaceClassLib">
36         <Version>1.0.0</Version>
37         <InterfaceClass Name="PorzessStartInterface">
38             <Attribute Name="StartAttribut" AttributeDataType="xs:dateTime" Unit="ZEIT">
39                 <DefaultValue>2016-08-20T15:30:44.2428373+02:00</DefaultValue>
40             </Attribute>
41             <InterfaceClass Name="FertigungsProzessStart"
42                 RefBaseClassPath="InterfaceClassLib/PorzessStartInterface" />
43         </InterfaceClass>
44     </InterfaceClassLib>

```

ABBILDUNG 19 AUTOMATIONML INTERFACECLASSLIBRARY

Rollenklassenbibliothek: Eine Rolle besitzt die Aufgabe der Elementtypisierung ähnlich zu den Interfaces innerhalb der Interfacebibliothek, welche eine Typisierung für Schnittstellen darstellt. Sie beinhaltet ebenfalls eine Versionsnummer, um Änderungen zu repräsentieren. Eine Rolle kann einem InternalElement oder einer System-Unit-Klasse zugewiesen werden. Dies erfolgt auf zwei möglichen Wegen. Die Rolle kann als Typ zugewiesen werden. Dies bedeutet das zugewiesene Element ist vom Typ dieser Rolle oder eine Rolle wird einem Element als Bedürfnis zugewiesen. Das Zuweisen von Rollen als Bedürfnis ist nur auf InternalElements erlaubt. Ein solches InternalElement repräsentiert einen Platzhalter für ein Element, dessen Typ die Bedürfnisrolle erfüllt. Rollen können hierbei weitere Rollen, Attribute und Instanzen von Interfaceklassen enthalten. Die *Abbildung 20* stellt die Beispielrollenklassenbibliothek „RoleClassLib“ dar. Diese besitzt die Version 1.0.0 und zwei Rollenklassen. Es handelt sich dabei um eine leere Rollenklasse „Role2“, welche keine Interfaces und Attribute enthält und der Rollenklasse „Role1“. Diese Rolle besitzt das Attribut „RollenAttribut“ und ein Interface, welches die Bezeichnung „Start“ trägt und eine Instanz von der Interfaceklasse

„ProzessStartInterface“ darstellt. „Role1“ könnte somit die Rolle eines Startpunktes von einem Prozess darstellen. Diese Rolle erbt über das „RefBaseClassPath“ Attribut von der Rolle „Role2“ und erhält somit eine „is a“ Relation.

```

48     <RoleClassLib Name="RoleClassLib">
49         <Version>1.0.0</Version>
50         <RoleClass Name="Role1">
51             RefBaseClassPath="RoleClassLib/Role2">
52                 <Attribute Name="RollenAttribut" />
53                 <ExternalInterface Name="Start">
54                     RefBaseClassPath="InterfaceClassLib/ProzessStartInterface" />
55             </RoleClass>
56         <RoleClass Name="Role2" />
57     </RoleClassLib>

```

ABBILDUNG 20 AUTOMATIONML ROLECLASSLIBRARY

System-Unit-Klassenbibliothek: Diese dient als letztes Teil der deklarativen Phase und wird für die Erstellung von System-Unit-Klassen genutzt. Hierbei wird eine solche Klasse als Art abstraktes Bauteil oder Produktbeschreibung gesehen, welche in der folgenden Instanz-Hierarchie instanziiert werden kann und dort wie in Programmiersprachen ähnlich als Objektinstanz einer Klasse agiert. System-Unit-Klassen können ähnlich zu Rolleninterfaces instanziiert werden und Attribute besitzen. Sie können aber zusätzlich weitere System-Unit-Klassen sowie Internal Elements besitzen, um hierdurch komplexe Objektstrukturen zu beschreiben. Die *Abbildung 21* stellt eine beispielhafte System-Unit-Klassenbibliothek dar, mit der Bezeichnung „SystemUnitClassLib“. Die aktuelle Version ist die 1.0.0 und sie besitzt die System-Unit-Klasse „FertigungsprozessBauteilA“, welche das Attribut „Dauer“ beinhaltet. Dieses bildet die benötigte Zeit eines Prozesses ab und besitzt den Datentyp „unsigned Long“, die Einheit „Sekunde“ und einen Default Value von „0“. Zusätzlich enthält es eine Interfaceinstanz „Start“ der Interfaceklasse „ProzessStartInterface“ und ein symbolisches InternalElement „BeschreibungAblauf, welches beispielhaft für ein Objekt mit tiefer beschreibender Intension steht. Es ist zu erkennen, dass jedes InternalElement hierbei eine ID erhält. Dies liegt daran, dass dieses immer die Instanz von etwas darstellt. Es kann sich hierbei um die konkrete Instanz einer System-Unit-Klasse handeln oder auch um das instanziierte Bedürfnis einer Rolle.

```

58     <SystemUnitClassLib Name="SystemUnitClassLib">
59         <Version>1.0.0</Version>
60         <AdditionalInformation>TEST</AdditionalInformation>
61         <SystemUnitClass Name="FertigungsprozessBauteilA">
62             <Attribute Name="Dauer">
63                 AttributeDataType="xs:unsignedLong" Unit="Sekunden">
64                     <DefaultValue>0</DefaultValue>
65             </Attribute>
66             <ExternalInterface Name="Start">
67                 RefBaseClassPath="InterfaceClassLib/ProzessStartInterface">
68                     ID="30958c77-5459-4153-b542-5105d9549fc7" />
69             <InternalElement Name="BeschreibungAblauf">
70                 ID="943f7487-971f-47ff-bd06-e13b0cc8c6fb" />
71             <SupportedRoleClass RefRoleClassPath="RoleClassLib/Role2" />
72         </SystemUnitClass>
73     </SystemUnitClassLib>

```

ABBILDUNG 21 AUTOMATIONML SYSTEMCLASSLIBRARY

Instanz-Hierarchie: Die Instanz-Hierarchie bildet die Instanz Phase von AutomationML ab und besteht aus hierarchisch strukturierter InternalElements. Jedes dieser Elemente kann Attribute, Interface Instanzen, zugewiesene Rollen und weitere InternalElements besitzen, wie innerhalb der System-Unit-Klassenbibliothek. Die Instanz-Hierarchie dient dazu, einen Übergang von der abstrakten Deklaration eines Objektes zu einem existierenden Instanz-Objekt zu repräsentieren. So wie ein produziertes Produkt das instanziierte Objekt einer Produktbeschreibung darstellt. Die innerhalb der *Abbildung 22* dargestellte Instanz-Hierarchie stellt somit mit dem „Fertigung_1041“ Objekt ein Instanz-Objekt der System-Unit-Klasse „FertigungsprozessBauteilA“ dar und befüllt z.B. das noch freie Feld „Value“ des Attributes „Dauer“ mit dem Wert „42“.

```

17     <InstanceHierarchy Name="InstanceHierachy">
18         <Version>1.0.0</Version>
19         <InternalElement Name="Fertigung_1041"
20             RefBaseSystemUnitPath="SystemUnitClassLib/FertigungsprozessBauteilA"
21             ID="11c3ea27-c69a-46ec-817c-97f26bf96ab0">
22             <Attribute Name="Dauer"
23                 AttributeDataType="xs:unsignedLong" Unit="Sekunden">
24                 <DefaultValue>0</DefaultValue>
25                 <Value>42</Value>
26             </Attribute>
27             <ExternalInterface Name="Start"
28                 RefBaseClassPath="InterfaceClassLib/PorzessStartInterface"
29                 ID="e46a774d-afba-46f8-b6eb-d404db2d6120" />
30             <InternalElement Name="BeschreibungAblauf"
31                 ID="3ac6b293-879a-4e26-b826-711bb8d1c418" />
32                 <SupportedRoleClass RefRoleClassPath="RoleClassLib/Role2" />
33             </InternalElement>
34     </InstanceHierarchy>

```

ABBILDUNG 22 AUTOMATIONML INSTANCEHIERARCHY

Für die vereinfachte Erstellung von AutomationML Dateien wird der AutomationML Editor zur Verfügung gestellt. Der Editor besitzt die Möglichkeit „aml“ Dateien auf Validität zu prüfen und eine zusätzliche GUI, welche für jede Klassenbibliothek und die Instanz-Hierarchie ein eigenes Teilfenster besitzt. *Abbildung 23* visualisiert die innerhalb dieser Arbeit genutzte Beispiel- „aml“-Datei, welche für die Beispiele dieses Kapitels genutzt und mit Hilfe dieses Werkzeuges erstellt wurden. Darin können vereinfacht neue Elemente in der jeweiligen Bibliothek erstellt werden und diese über „Drag and Drop“ innerhalb der Instanz-Hierarchie bestehenden Objekten zugewiesen oder dort als neue InternalElements instanziiert werden. Die Rechtseite besitzt eine Konfigurationsleiste, welche für die Konfiguration einzelner Elemente zuständig ist. Wählt man ein Element aus, kann man z.B. innerhalb dieser Leiste Attribute bearbeiten oder dem Element einen neuen Namen geben.

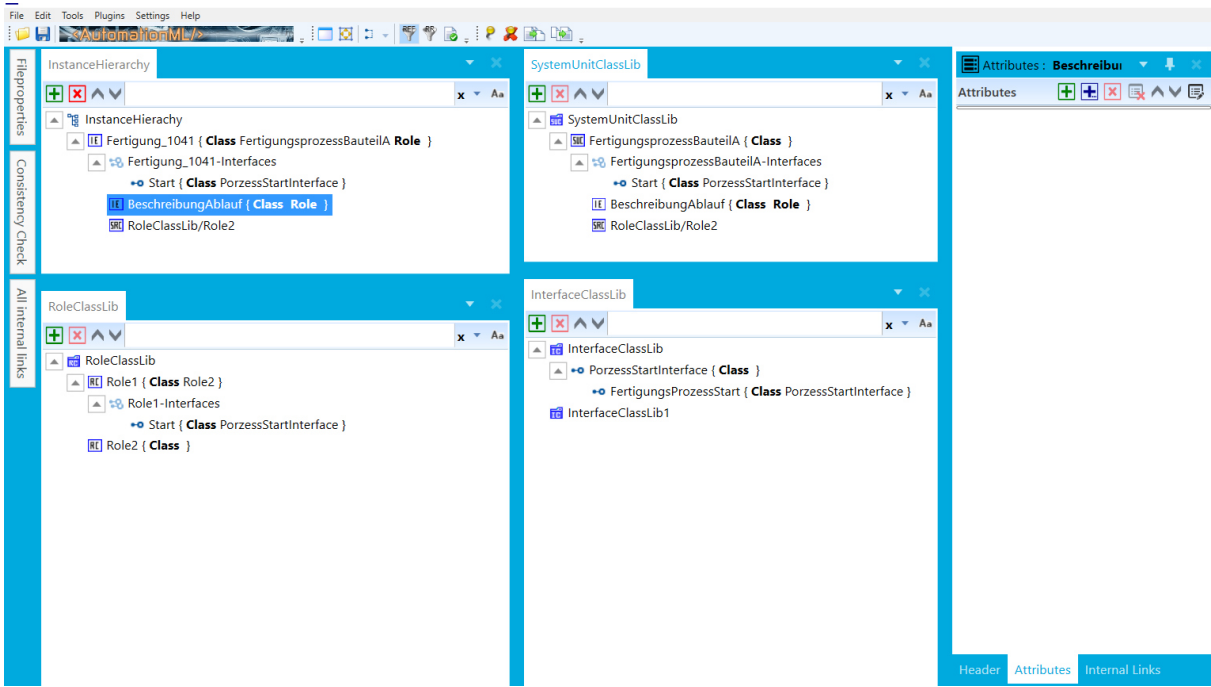


ABBILDUNG 23 AUTOMATIONML EDITOR

2.3 INDUSTRIE 4.0 WERTSCHÖPFUNGSKETTE

Die Wertschöpfungskette [1] spiegelt das Potential wieder, welches innerhalb der neuen industriellen Revolution Industrie 4.0 steckt. Eine Wertschöpfungskette beschreibt alle Verzweigungen einer industriellen Fertigung, welche direkten Einfluss aufeinander nehmen und somit potentielle Informationen liefern, um diese zu verbessern. Die Abbildung 24 bildet alle von der VDI/VDE-GMA-Fachausschuss „Industrie 4.0“ definierten Wertschöpfungsketten innerhalb von Industrie 4.0 ab. Es werden hierbei 4 Hauptwertschöpfungsketten spezifiziert, welche im Folgenden näher beschrieben werden.

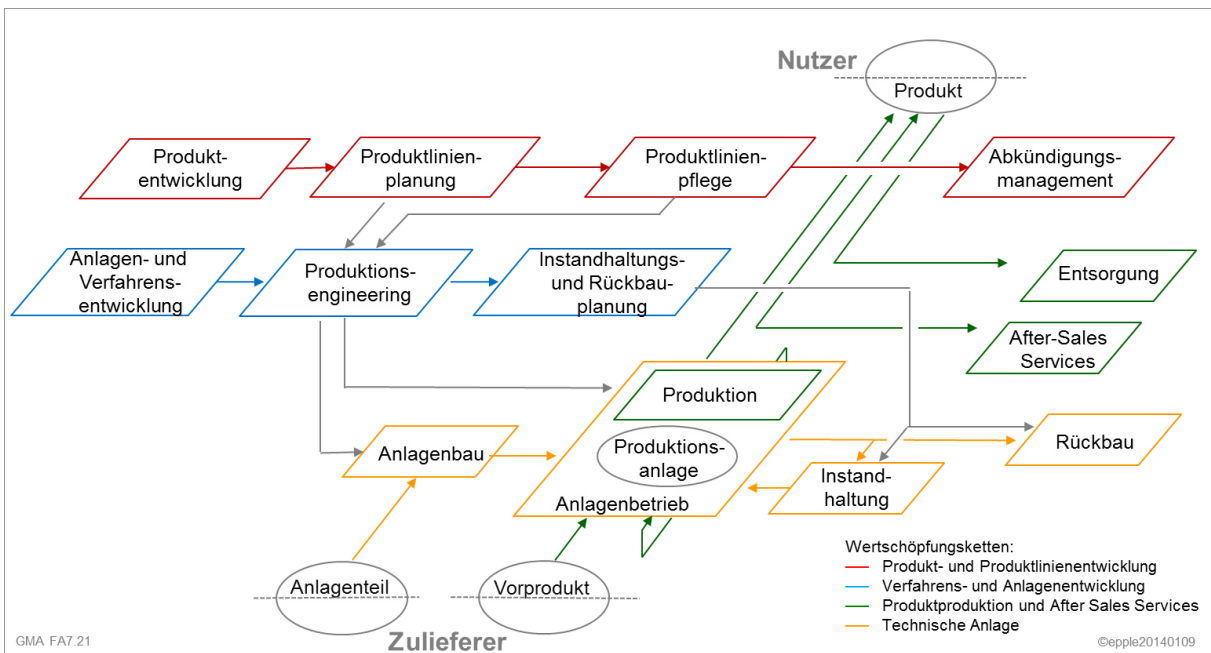


ABBILDUNG 24 INDUSTRIE 4.0 WERTSCHÖPFUNGSKETTEN [1]

Produkt- und Produktlinienentwicklung dient der Entwicklung von neuen Produkten eines Unternehmens und setzt sich aus multiplen Schritten zusammen. Nach Findung einer Produktidee muss diese innerhalb der Produktentwicklung zu einem spezifizierten Produkt heranreifen. Das hieraus entstandene Produkt wird im nächsten Schritt der Produktlinienplanung zu einer Vielzahl von Produkten, welche ökonomisch oder funktional optimiert wurden. Die darin enthaltenen Dokumente beschreiben die Produktionslinie eines Produktes wie z.B. Produktionsvorschriften, Ablaufpläne oder Normen. Im Anschluss werden diese Produkte während ihrer Laufzeit als Produkt eines Unternehmensportfolios weiter innerhalb des Produktlinienmanagements gepflegt. In der letzten Phase, dem Abkündigungsmanagement, wird das jeweilige Produkt aus einem Unternehmensportfolio entfernt. Diese Wertschöpfungskette, separat abgebildet in *Abbildung 25*, generiert produktbeschreibende Dokumente. Es handelt sich hierbei um Daten für die Beschaffenheit sowie optische Erscheinung, dargestellt über Produktspezifikationen und CAD – Zeichnungen sowie Dokumente für die Spezifikation von Fertigungsbedingungen.



ABBILDUNG 25 WERTSCHÖPFUNGSKETTE PRODUKT- UND PRODUKTLINIENENTWICKLUNG [1]

Verfahrens- und Anlagenentwicklung beinhaltet die Planung einer Anlage innerhalb der ersten Phase der Verfahrensentwicklung. Eine Produktionsanlage wird nicht für ein einzelnes Produkt entwickelt, sondern für eine Kategorie von Produkten. So sind in der Verfahrensentwicklung die Verfahren die eine Produktionsanlage bereitstellt, ein Schwerpunkt dieser Entwicklung. Im Anschluss wird durch das Produktionsengineering die Produktion eines Produktes oder einer Produktlinie auf der zuvor geplanten Anlage realisiert, welche z.B. die Realisierung von einer Qualitätssicherung beinhalten kann. Dabei werden teilweise Dokumente aus der Produktlinienplanung und dem Produktlinienmanagement genutzt. Danach wird innerhalb der Instandhaltungsplanung ein Wartungsplan erstellt, der alle Wartungsarbeiten und ihre Bedingungen definiert. Dabei kann es sich um Definitionen des Wartungsprozesses, des Wartungspersonals oder zeitliche Beschränkungen von Wartungsarbeiten handeln. Die letzte Phase dieser Wertschöpfungskette ist die Stilllegungsplanung. Diese beinhaltet einen Prozess für das Auslaufen lassen einer Produktion, den Abbau einer Anlage sowie Besonderheiten für die Entsorgung von Anlagenbauteilen. Die gesamte Wertschöpfungskette ist isoliert in *Abbildung 26* dargestellt. Die darin entstehenden Dokumente umfassen beschreibende Dokumente für die genutzten Fertigungsmaschinen, die Beschreibung der Verbindung dieser Maschinen und den Fertigungsprozess für welchen diese genutzt werden. Zusätzlich entstehen Vorlagen für Wartungsdokumente, Wartungsspezifikationen und Dokumente für die Stilllegung einer Anlage.

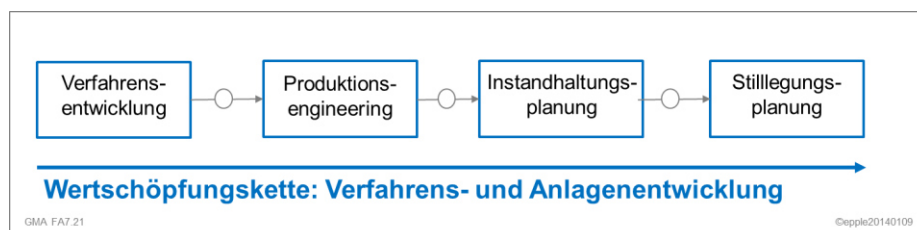


ABBILDUNG 26 WERTSCHÖPFUNGSKETTE VERFAHRENS- UND ANLAGENENTWICKLUNG [1]

Produktproduktion und After Sales Services beinhaltet alle Prozesse, welche für die Erschaffung einer Produktinstanz zuständig sind und damit verbundene Dienstleistungen. Zudem können darin auch weitere Produkte für die Betreuung oder Instandhaltung dieser Instanz enthalten sein. Die initiale Phase, die industrielle Produktion, erzeugt die Instanziierung des innerhalb des Produktionsengineering spezifizierten Verfahrens für die Herstellung des Produktes. So entstehen für spezifizierte Soll – Werte dazugehörige reale Ist – Werte aus der Produktion. Die anschließende Nutzungsphase durch den Kunden gehört nicht direkt zu der hier beschriebenen Wertschöpfungskette, da hierbei kein direktes Feedback stattfindet. Werden zusätzliche After Sales Services begleitend für ein Produkt angeboten, können diese wieder neue Informationen bezüglich des produzierten Produktes generieren. Ein solcher After Sales Service kann die Wartung, Instandhaltung, Inbetriebnahme, Personalfortbildungen, Consulting und die Überwachung des ausgelieferten Produktes beinhalten. Diese liefern zusätzlich Informationen über Probleme des Kunden mit dem Produkt, Eigenschaften des Produktes, die nicht wie spezifiziert agieren oder auch potentielle Erweiterungen der Produktlinie. Die Endphase dieser Wertschöpfungskette ist eine optionale Entsorgung des Produktes. Dies ist eine spezielle Form eines After Sales Service und gibt die Chance für eine tiefere Analyse eines Produktes, welches seinen Lebenszyklus durchlaufen hat. Die Verbindungen der einzelnen Phasen werden in *Abbildung 27* separat dargestellt.

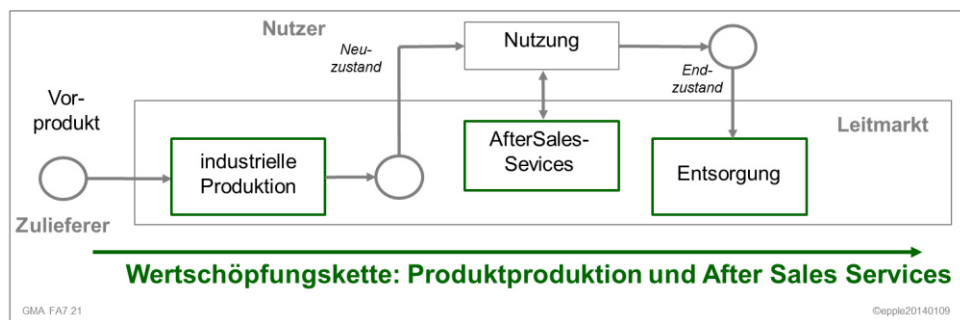


ABBILDUNG 27 WERTSCHÖPFUNGSKETTE PRODUKTPRODUKTION UND AFTER SALES SERVICES [1]

Technische Anlage stellt alle Prozesse dar, welche mit der Produktion eines Produktes auf einer Fertigungsanlage korrelieren. Zu Beginn wird durch die Phase des Anlagenbaus, die innerhalb der Anlagenentwicklung geplanten Produktionsanlage instanziiert. Darauf folgt eine weitere Phase der industriellen Produktion. Diese betrachtet dieselben Prozesse wie die industrielle Produktion innerhalb der *Wertschöpfungskette Produktproduktion und After Sales Services*, betrachtet diese jedoch aus der Sicht der Produktionsanlage. Das bedeutet, dass hierbei nicht die Eigenschaften betrachtet werden

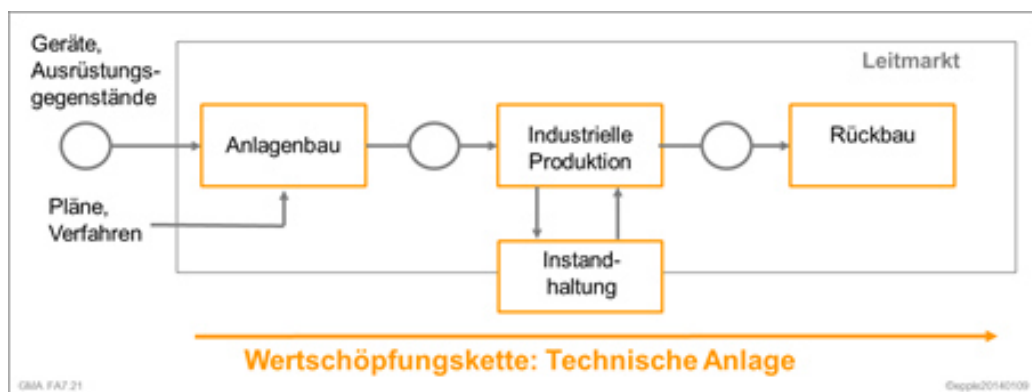


ABBILDUNG 28 WERTSCHÖPFUNGSKETTE TECHNISCHE ANLAGE [1]

welche für die Qualität des Produktes ausschlaggebend sind, sondern jene die für die Qualität der Produktionsanlage stehen. Dies könnte z.B. der Abnutzungsgrad eines Schleifwerkzeuges sein. Eine parallel oder auch im Wechsel hierzu auszuführende Phase ist die Instandhaltung der Produktionsanlage, welche Instanzen von Instandhaltungsprotokollen generiert. Die Beschreibungen dieser Protokolle stammen aus der Instandhaltungsplanung. Das Ende eines Lebenszyklus einer Produktionsanlage leitet die Rückbau Phase ein. Der Ablauf der Rückbau Phase wird in der Stilllegungsplanung definiert und stammt aus der *Wertschöpfungskette Verfahrens- und Anlagenentwicklung*. Diese Zusammenhänge werden in der *Abbildung 28* illustriert.

2.4 RAMI 4.0

Das Referenzarchitekturmodell für Industrie 4.0 (RAMI 4.0) stellt ein Konsensprodukt von deutschen Industrieteilnehmern der Fertigungsindustrie und Verbänden dar. Es ist ein Teilprodukt des VDI/VDE GMA Fachausschusses 7.21 „Industrie 4.0“ [3] [25]. Das Ziel von RAMI 4.0 ist eine einheitliche Architektur zu spezifizieren, welche die gesamte Industrielandschaft abbildet und die Möglichkeit bietet, über eine einheitliche Komponentendefinition, Elemente der Industrie 4.0 plattformübergreifend darzustellen. RAMI4.0 stellt die Anforderungen für diese Arbeit bereit. Es werden im Folgenden die Referenzarchitektur, Servicearchitektur, die I4.0-Verwaltungsschale und die I4.0-Komponente von RAMI4.0 präsentiert.

2.4.1 REFERENZARCHITEKTUR

Die Referenzarchitektur ist ein dreidimensionales Modell, jede der drei Dimensionen stellt eine Perspektive auf die Industrie 4.0 dar. Das Modell ist angelehnt an das Smart Grid Architekturmodell (SGAM) [26]. *Abbildung 29* visualisiert eine Übersicht über das gesamte Modell. Die Abbildung macht deutlich, dass die erste **Layer**-Perspektive das Modell in verschiedene Schichten aufteilt. Diese Schichten werden über die **Hierarchie – Levels**, welche aus den Normen IEC 62264 und IEC 61512 entnommen wurden, in verschiedene Ebenen, bezüglich der Granularität von Komponenten innerhalb einer Industrieanlage, aufgeteilt. Die letzte Achse bildet den **Value Stream** oder auch den **Lebenszyklus** von solchen Komponenten nach der Norm IEC 62890 ab. So ist das Ziel von RAMI4.0 Komponenten innerhalb von Industrie 4.0 in einem Modell abzubilden. Im Folgenden werden die drei Perspektiven des Modells präsentiert.

Die **Layer**-Achse teilt das Modell in 6 Schichten auf. Die oberste **Geschäftsschicht** (Business Layer) dient dazu Geschäftsprozesse oder Geschäftsmodelle abzubilden, die von einem Unternehmen verfolgt oder vollzogen werden. Dabei sind keine explizit genutzten Technologien oder Plattformen Teil dieser Schicht. Sie enthält zusätzlich Informationen bezüglich der Verbindung von Geschäftsprozessen, rechtlichen Rahmenbedingungen und Geschäftsbedingungen, welche für das vollständige Ausführen eines Geschäftsprozesses nötig sind. Die darunterliegende **Funktionsschicht** (Function Layer) beschreibt Dienste, die Teil eines oder mehrerer solcher Geschäftsprozesse sind. Trotz dessen, dass diese Dienste Teil eines Geschäftsprozesses darstellen, sind sie nicht Teil der Geschäftsschicht, da diese nur zur Realisierung des Prozesses genutzt werden, selbst aber keinen gesamten Geschäftsprozess darstellen. Benötigen solche Dienste Informationen, so beziehen sie diese aus der **Informationsschicht** (Information Layer). Diese Schicht bildet die Daten eines Unternehmens ab, welche über ein wohldefiniertes Datenmodell und einheitliche Schnittstellen zugänglich sein sollen. Eine **Kommunikationsschicht** (Communication Layer) sorgt über die Verwendung eines einheitlichen Datenformates für die nötigen Daten innerhalb der Informationsschicht. Der Übergang von Informationen aus der **Asset-Schicht** (Asset Layer) in die Kommunikationsschicht erfolgt über die **Integrationsschicht** (Integration Layer). Die Asset-Schicht beinhaltet hierbei alle realen Objekte. Diese

Objekte werden innerhalb von Industrie 4.0 als Assets [3] bezeichnet und können von Kleinstbauteilen bis hin zu menschlichen Individuen reichen. Wie in der Relation der Schichten zueinander angedeutet, ist die Kommunikation nur zwischen benachbarten Schichten vorgesehen. Das Ziel soll es sein, dass Schichten eine geringe bis lose Kopplung untereinander aufweisen, um so einen hohen Grad an Flexibilität zu erreichen.

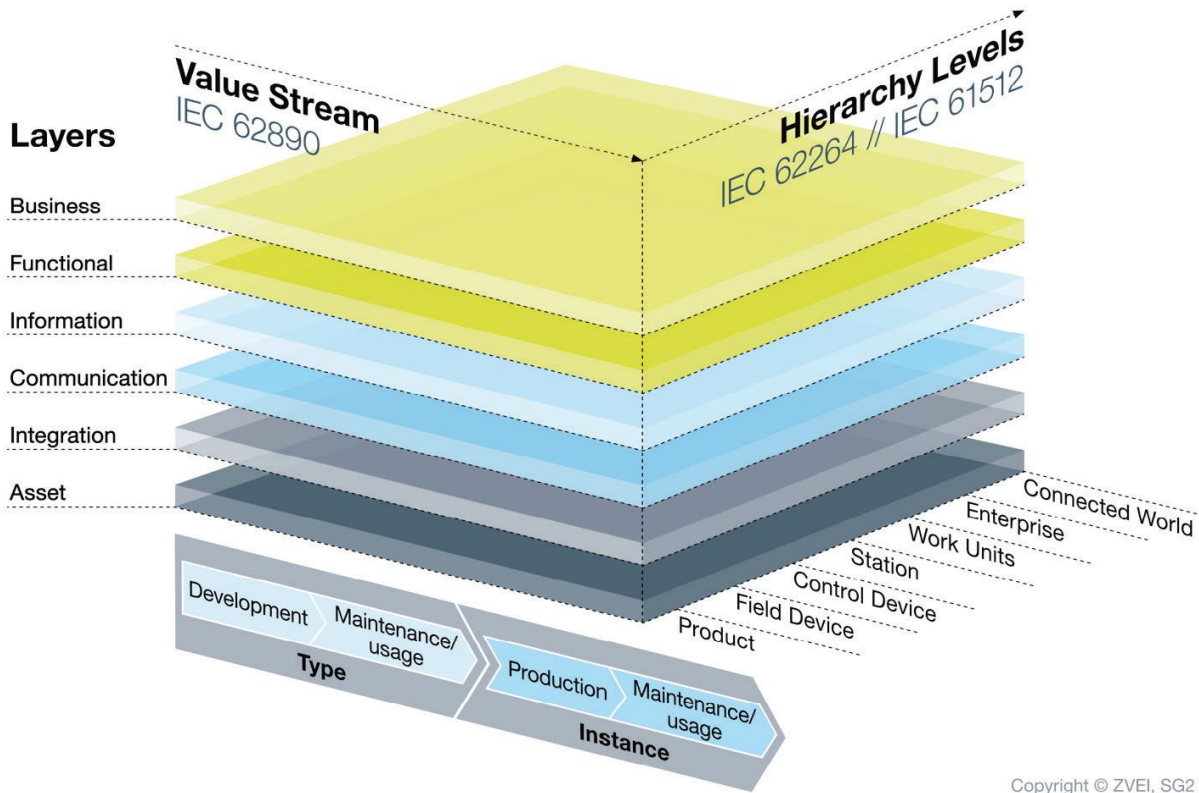


ABBILDUNG 29 RAMI4.0 ARCHITEKTUR [3]

Der **Lebenszyklus** oder auch **Value Stream** von Objekten spielt für die Datenhaltung eine wichtige Rolle. Dieser besitzt wie in *Abbildung 30* dargestellt eine zwei Phasenmentalität. So wird jedes Objekt, ob es sich um ein Produkt, einen Auftrag oder eine gesamte Produktionsfabrik handelt, in die Phasen **Typ** und **Instanz** aufgeteilt. Die startende Typ-Phase besitzt einen deklarierenden Charakter, denn es wird innerhalb dieser Phase das Objekt beschrieben ohne es real zu erzeugen. Alle Dokumente die das Objekt beschreiben gehören zu dieser Phase. Dies reicht von der einfachen Handskizze bis hin zur CAD, CAM oder CAP Zeichnung. Im Anschluss wird das deklarierte Objekt innerhalb der Instanz-Phase ein oder mehrfach instanziiert. Dies erfolgt meist durch die Produktion eines Objektes. Das Potential von Industrie 4.0 besteht darin den Lebenszyklus in all seinen Komponenten, durch eine Vernetzung von Typ- und Instanz-Phase, positiv zu beeinflussen. Dies kann durch ein direktes Feedback aus der Instanz-Phase geschehen, um z.B. Dokumente aus der Type-Phase zu verbessern, welche wiederum verbesserte neue Objekte in der Instanz-Phase erzeugen oder durch das angestrebte einheitliche Datenmodell Fehlerquellen ausfindig machen.

Die **Hierarchieschicht** spiegelt die verschiedene Granularität innerhalb der Produktion wieder. Ist ein Element eindeutig identifizierbar, dann kann es von der gesamten Welt des Internets oder einzelnen verbunden Unternehmen bis hin zu produzierten Produkten oder einzelnen Field Devices, im Sinne von IoT-Geräten, herunter gebrochen werden. Diese verschiedenen Granularitäten spiegeln wiederum unterschiedliche Kontexte wieder, in welchen die darüber abgebildeten Elemente in Verbindung stehen.

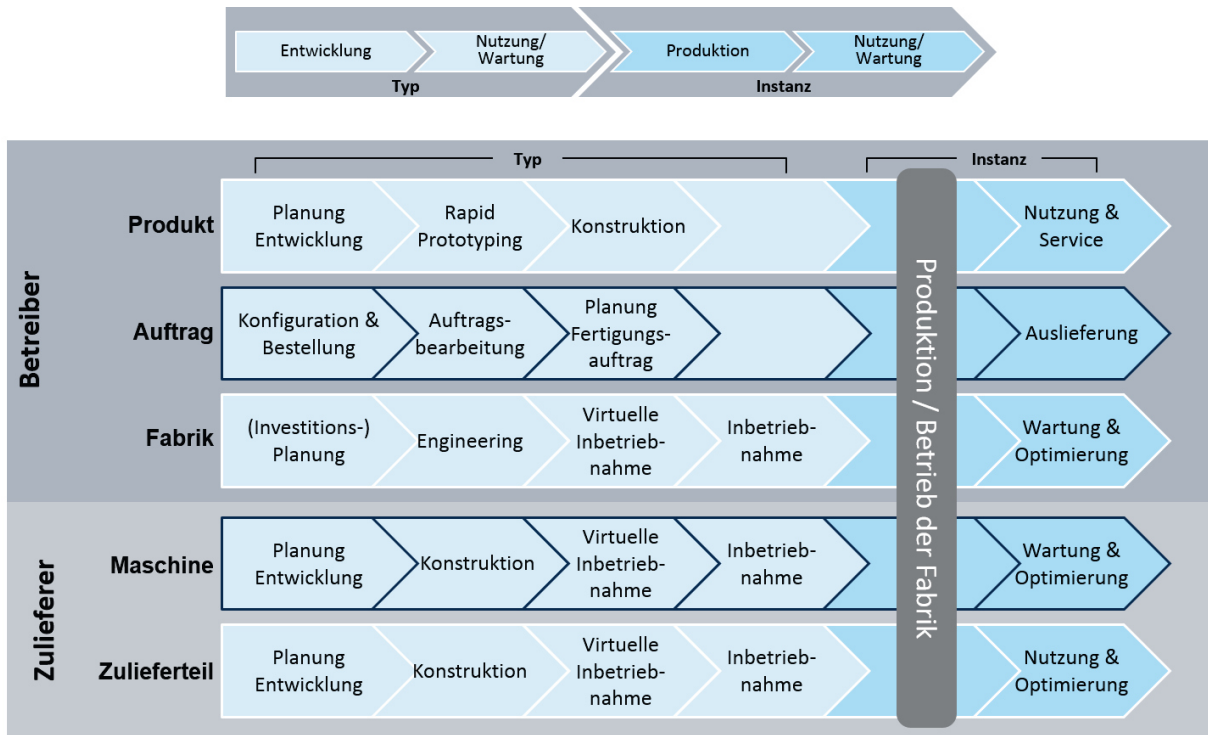


ABBILDUNG 30 LEBENSZYKLUS [3]

2.4.2 I4.0 KOMPONENTE

Eine I4.0-Komponente [3] [2] stellt nach der Definition des RAMI4.0 eine digitale Repräsentation von verschiedenen Elementen dar, welche Teil der Produktion sind. Diese Elemente, welche in Industrie 4.0 und im Folgenden innerhalb dieser Arbeit als Assets bezeichnet werden, werden innerhalb einer I4.0-Komponente von einer Verwaltungsschale gemanagt. Dies beinhaltet, wie in *Abbildung 31* dargestellt, die Möglichkeit, dass ein Asset, wie z.B. eine Produktionsmaschine, aus mehreren Asset besteht, wie z.B. Sensoren und Motoren. Diese Assets sollen innerhalb einer I4.0-Komponente virtuell repräsentiert werden. Zusätzlich können I4.0-Komponenten mithilfe einer I4.0 konformen Kommunikation miteinander verbunden sein. Hierfür sollen diese über ein einheitliches Datenformat Funktionalitäten sowie Informationen bereitstellen und nutzen können.

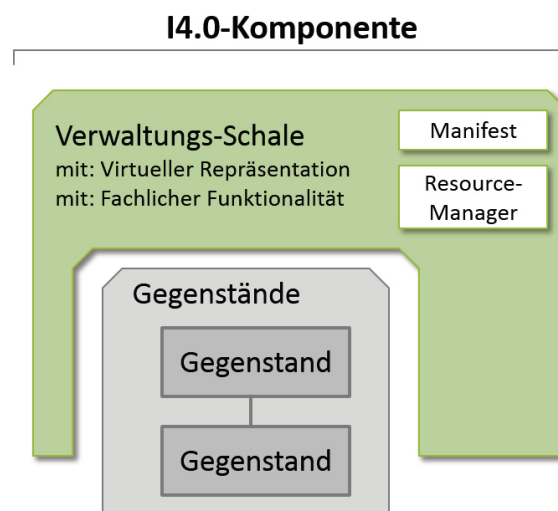


ABBILDUNG 31 I4.0-KOMPONENTE [3]

2.4.3 VERWALTUNGSSCHALE

Die Verwaltungsschale [3] realisiert die virtuelle Repräsentation von Assets, sowie ihrer Relation zueinander. Sie bietet zusätzlich Funktionalitäten an, welche eine direkte Weiterleitung spezieller Asset-Funktionen darstellen können oder eine Kombination aus Asset-Funktionalitäten und Asset-Informationen, so kann z.B. eine Verwaltungsschale Funktionen einer Bohrmaschine bereitstellen. Dies soll durch ein direktes Mapping möglich sein, in welchem die Daten des Assets in das Datenmodell der Verwaltungsschale überführt werden. Eine solche Überführung kann durch die Kombination eines Assets geschehen, welches einen Standard für Bohrungen definiert und mithilfe dessen Daten aus der Kombination der Bohrmaschinen-Asset Daten eine vereinfachte abstrakte Funktion „Bohrung“ nach außen bereitgestellt werden kann. Die zweite dieser Varianten würde es somit vereinfachten Standards in die digitale Repräsentation der Industrie mit einzubringen. Dies soll, wie in *Abbildung 32* dargestellt, nicht durch das Ersetzen der aktuell verwendeten Daten durch ein einziges Datenmodell geschehen.

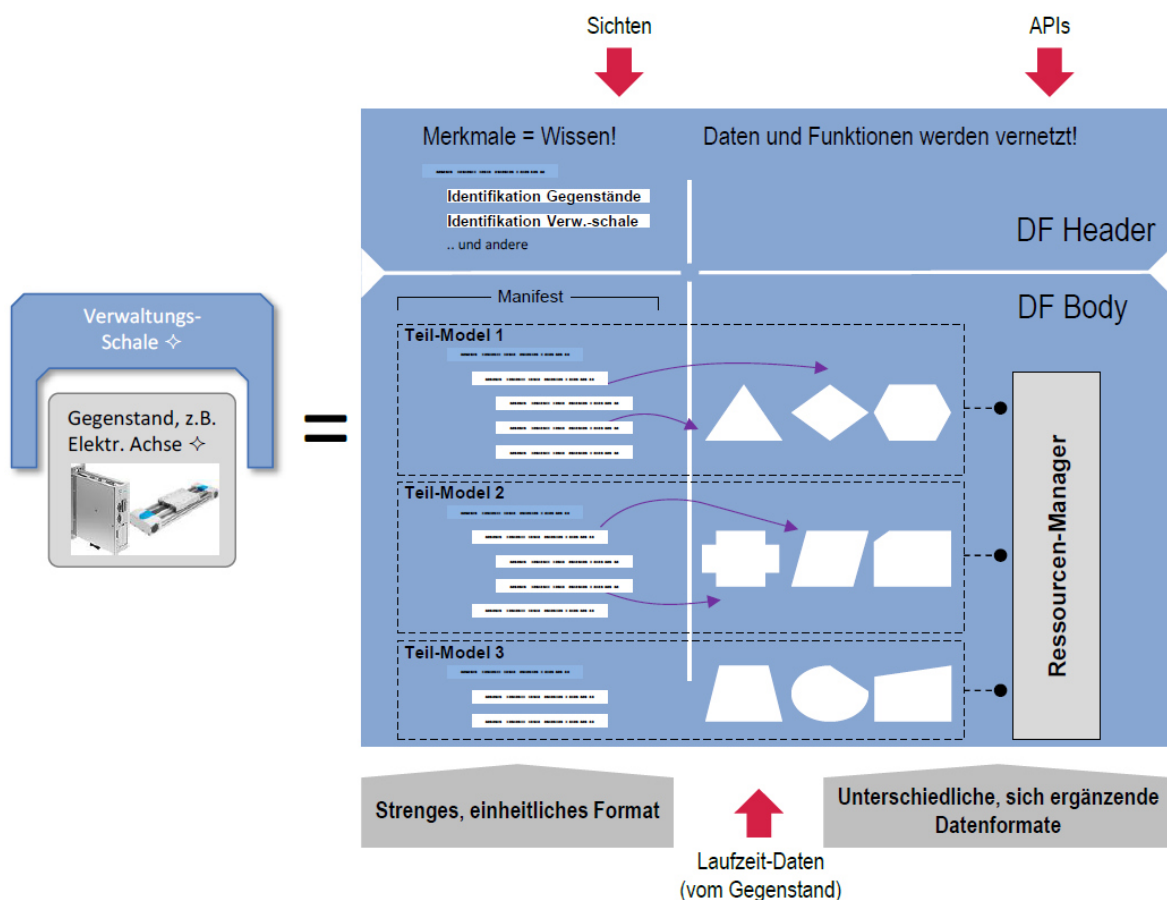


ABBILDUNG 32 VERWALTUNGSSCHALE-DATENFORMAT [3]

Das angestrebte Datenmodell der Verwaltungsschale soll dies zwar ermöglichen, das Hauptziel ist es jedoch die Daten aus verschiedenen Quellen zu abstrahieren und diese Abstraktion innerhalb des Datenmodells darzustellen. Es sollen dabei Verlinkungen entstehen, welche einen Rückschluss auf die Quelldaten erlauben. Diese Abstraktion sowie die Quelldaten sollen von einem Ressourcenmanager überwacht werden. Zum Zeitpunkt dieser Arbeit wurden für die Verwaltungsschale die in *Tabelle 1* gelisteten Funktionen definiert, diese sollen von außerhalb erreichbar sein. Zudem muss eine Verwaltungsschale die innerhalb der *Tabelle 2* aufgeführten Eigenschaften erfüllen und die Möglichkeit bieten, wie in *Abbildung 33* visualisiert, eine operative Hierarchie bezüglich weiterer

Verwaltungsschalen zu leisten. Dies bedeutet, dass eine Verwaltungsschale die Funktionalität von mehreren Verwaltungsschalen und Assets für eigene Funktionalitäten aggregieren oder mit Bezug auf Aspekte der Sicherheit den Zugriff limitiert weiterleiten kann. Dies wird in *Abbildung 33* beispielhaft dargestellt, in dem eine als „obere“ I4.0 Kommunikationsnetzwerk bezeichnete Infrastruktur nur über die dazwischenliegende Verwaltungsschale auf das I4.0 Kommunikationsnetzwerk „unten“ Zugriff erlangt. Dieser indirekte Zugriff, dargestellt über die gestrichelte Linie mit der Ziffer 2, erfolgt über die Schnittstellen der dazwischenliegenden Verwaltungsschale, markiert mit der Ziffer 1. Diese Trennung soll es erlauben den Zugriff des Netzwerkes „oben“ auf I4.0-Komponenten des Netzwerkes „unten“ zu kontrollieren und zu überwachen.

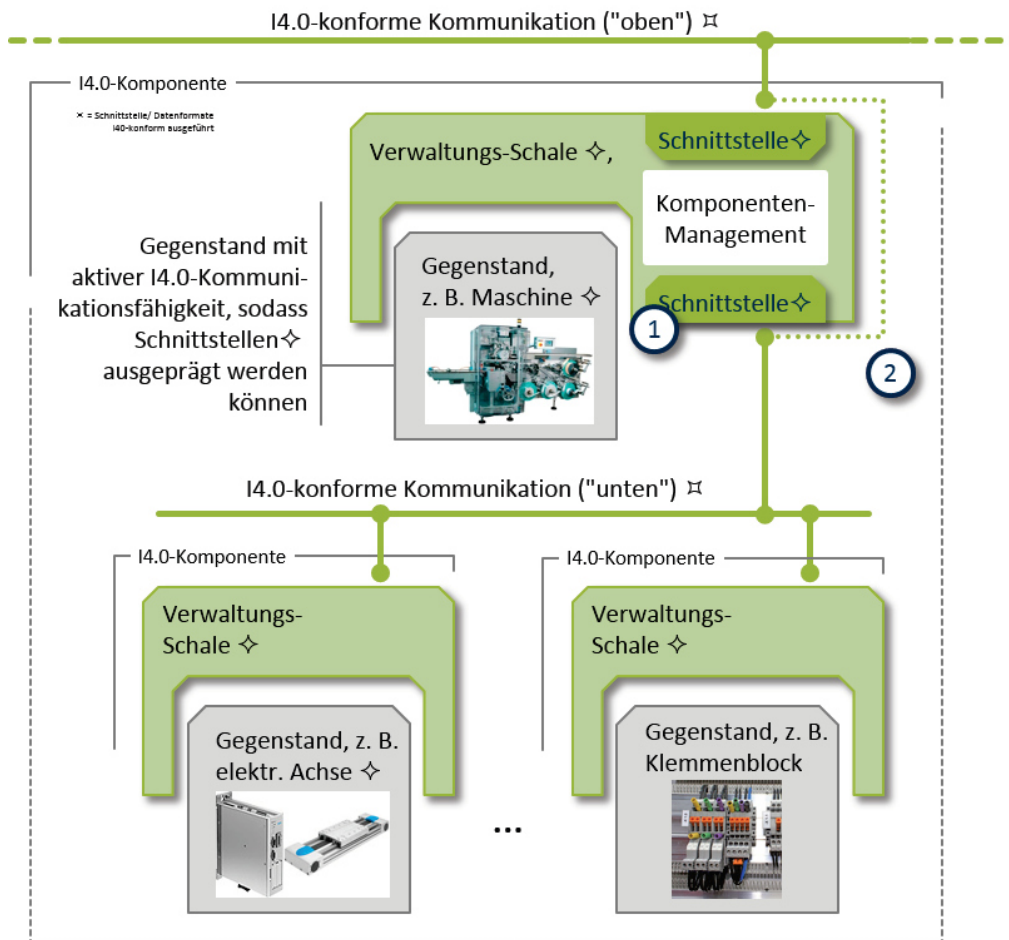


ABBILDUNG 33 VERWALTUNGSSCHALENVERNETZUNG [3]

TABELLE 1 VERWALTUNGSSCHALEN FUNKTIONEN [3]

Funktionalität	Beschreibung
Lesen	Diese Funktion ermöglicht es Werte aus dem Datenmodell zu lesen. Dies beinhaltet Objekte sowie einzelne Attribute dieser Objekte, wie z.B. die aktuelle Wertebelegung einer OPC UA Node Repräsentation.
Schreiben	Das Schreiben auf Objekte innerhalb des Datenmodells soll möglich sein, ebenso das Schreiben auf einzelne Attribute des Objektes bei Bedarf.
Create	Das Erstellen neuer Objekte innerhalb des Datenmodells einer Verwaltungsschale nach dem I4.0 konformen Datenmodell soll durchführbar sein.
Delete	Für die Situation, in welcher ein nicht mehr benötigtes Datenobjekt nicht von einer internen Routine gelöscht wird, soll es eine Funktion geben um diese manuell zu entfernen.
Abonnieren	Eine Verwaltungsschale soll es ermöglichen die Werte eines Datenobjektes innerhalb des Datenmodells zu abonnieren. Zu dieser Funktion gehört auch das Veröffentlichen dieser Abos zum Empfänger angegebenen Endpunkt.
Browse	Das gesamte Modell kann ausgelesen werden, um alle zur Verfügung stehenden Daten einer Verwaltungsschale einzusehen. Die Browse-Funktion stellt diese Funktionalität bereit und ermöglicht eine gezielte Suche nach den gesuchten Daten.
Method Call	Für die Deklaration eigener Funktionalitäten soll ein abstrakter Methodenaufruf unterstützt werden. Dies setzt voraus, dass die Verwaltungsschale und deren Datenmodell in der Lage ist solche Funktionen abzubilden. Hierdurch sollen die deklarierten Funktionen nutzbar sein. Method Calls bilden die Applikation-Services ab.

TABELLE 2 VERWALTUNGSSCHALE EIGENSCHAFTEN [3]

Eigenschaft	Beschreibung
Identifizierbarkeit	Eine Verwaltungsschale muss innerhalb eines Netzwerkes eindeutig identifizierbar sein. Dies trägt dazu bei, dass eine Kommunikation und Referenz zwischen I4.0-Komponenten möglich ist.
I4.0 Kommunikation	Eine I4.0 konforme Kommunikation muss unterstützt werden. Um welche Kommunikationstechnologie es sich handeln wird ist zum Zeitpunkt dieser Arbeit noch unklar und soll noch bestimmt werden.
I4.0 Zustände und Dienste	Es werden die innerhalb von <i>Tabelle 1</i> beschriebenen Funktionen unterstützt. Eine I4.0 konforme Repräsentation der Zustände von Objekten innerhalb der Verwaltungsschale soll ermöglicht werden.

Virtuelle Beschreibung	Über die Nutzung eines I4.0 Datenmodells werden Informationen abgebildet, welche den aktuellen Status, Asset-Informationen sowie bereitgestellte Funktionen darstellen. Dies erlaubt eine einheitliche Interpretation von Informationen.
I4.0 Semantik	Die Semantik der innerhalb der virtuellen Beschreibung dargestellten Objekte sowie deren Verwendung folgen einer definierten I4.0 Semantik, diese sind zum Zeitpunkt dieser Arbeit noch nicht definiert.
Security und Safety	Eine Verwaltungsschale muss für die Datenhaltung und deren Zugriff einen angemessenen Schutz bieten und zusätzlich die Einbettung von Schutzmechanismen ermöglichen.
QoS	Die bereitgestellten Funktionen werden als Service umgesetzt und erfüllen ihre definierte QoS Beschränkungen. Zudem muss eine Verwaltungsschale sowie ihr Datenmodell die Definition solcher QoS Attribute unterstützen.

2.4.4 SERVICEARCHITEKTUR

Die Servicearchitektur soll die Rahmenbedingungen für Dienstleistungen der verschiedenen RAMI4.0 Layer definieren. Sie befindet sich zum aktuellen Stand dieser Arbeit noch in der Definitionsphase. Sie beinhaltet die in **Fehler! Verweisquelle konnte nicht gefunden werden.** visualisierten vier Servicebereiche, welche sich über die RAMI4.0 Layer Funktionale-, Informations- und Kommunikationsschicht erstreckt. Dies erfolgt mit den **Plattform-** und **Applikationsdiensten** in der Funktionsschicht, **Informationsdienste** in der Informationsschicht und den **Kommunikationsdiensten** in der Kommunikationsschicht. **Plattformdienste** erstrecken sich über die Bereitstellung von Funktionen, welche benötigt werden, um Applikationen auf einer Plattform zu betreiben.

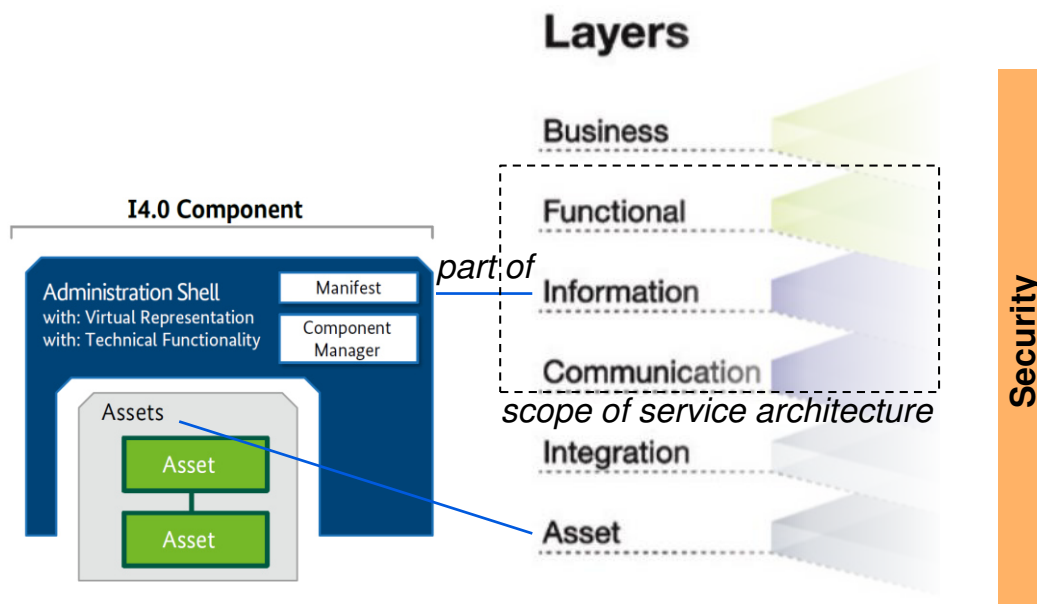


ABBILDUNG 34 SERVICEARCHITEKTURMODELL¹

¹ Quelle: VDI/VDE GMA FA 7.21 Whitepaper Status Report „Industrie 4.0 Service Architecture Basic Concepts for Interoperability“ [In Bearbeitung Stand vom 23.04.2016]

Diese ähneln den Paas-Diensten von Cloud Anwendungen. **Applikationsdienste** überlappen sich mit der Geschäftsschicht, da diese einzelne Anwendungen darstellen, welche Funktionen auf einer oder mehrere Plattformen zur Verfügung stellt. Diese Funktionalitäten können Dimensionen erreichen, in welchen ein gesamter Geschäftsprozess durchgeführt und überwacht wird. **Informationsdienste** stellen Daten aus der Informationsschicht bereit. Diese beziehen sie aus der Kommunikationsschicht, welche diese wiederum mit Hilfe der Kommunikationsdienste anbietet. Plattformdienste und Applikationsdienste bauen auf den Informationsdiensten auf. Sie benötigen diese Daten entweder für das Management einer Plattform oder für Interaktionen innerhalb einer Anwendung. Jeder dieser verschiedenen Servicebereiche soll untereinander eine lose Kopplung besitzen, um so die jeweiligen zur Verfügung gestellten Dienste jederzeit erweiterbar und austauschbar zu halten. Es wird in der momentanen Situation eine serviceorientierte Architektur [27] (SOA) angestrebt, mit welcher eine lose Kopplung erzielt werden soll.

3 KONZEPTION

In diesem Kapitel wird ein abstraktes Datenmodell erstellt. Hierfür werden aus dem Stand der Technik und den Grundlagen die Anforderungen an das Modell definiert. Im Anschluss wird die Herleitung dieses Modells erläutert und abschließend das Datenmodell vorgestellt.

3.1 ANFORDERUNGEN AN DAS DATENMODELL

In der folgenden *Tabelle 3* werden die Anforderungen an das Datenmodell zusammengefasst. Diese resultieren aus *Kapitel 2.4*.

TABELLE 3 ANFORDERUNGEN DATENMODELL

Anforderung	Beschreibung
Plattformunabhängigkeit	Das angestrebte Datenmodell soll universell einsetzbar sein. Es soll in der Lage sein auf aktuell genutzten Plattformen betrieben zu werden, sowie durch keine Abhängigkeiten zu diesen auf weitere Plattformen portierbar sein.
SOA	Die Interaktion mit Funktionalitäten einer Schicht des RAMI4.0 sollen einer serviceorientierten Architektur folgen. Es ist hierbei nicht nötig einen Service-Bus oder eine Service-Registry mit abzubilden. Die Funktionen einer Schicht sollen somit in Diensten gekapselt werden. Dies folgt aus der Servicearchitektur, welche beschrieben wurde in <i>Kapitel 2.4.4</i> .
Lose Kopplung	Es soll eine lose Kopplung zwischen diesen Diensten gehalten werden, welche über eine Schicht des RAMI4.0 Modells hinausgehen. Dies folgt aus der Servicearchitektur, welche beschrieben wurde in <i>Kapitel 2.4.4</i> .
Wiederverwertbarkeit	Modellierte Assets und Funktionalitäten, welche einen Prozess oder Dienst von Assets oder I4.0-Komponenten darstellen, sollen wiederverwertbar sein. Wenn ein Asset eine weitere Instanz eines bestimmten Typs darstellt, muss auch das Modell für dieses Asset valide sein. Dies bezieht sich auf den Lebenszyklus von Elementen innerhalb des RAMI4.0 Modells, welche in <i>Kapitel 2.4.1</i> präsentiert wurden.
Keine Technologieabhängigkeiten	Die genutzten Technologien, welche für die Umsetzung des SOA Ansatzes genutzt werden, sind noch nicht bestimmt. Daher sollte nicht nur das Modell und dessen genutztes Datenformat plattformunabhängig sein, sondern auch die Beschreibung von Diensten sollte sich nicht technologiespezifischen Beschreibungen bedienen und somit ein Mapping auf die größtmögliche Auswahl an Technologien erlauben.
I4.0-Komponentendatenmodell	Das angestrebte Datenmodell soll als Modell für die Verwaltungsschale von I4.0-Komponenten dienen. Daher muss das Modell in der Lage sein die Funktionen und Eigenschaften von I4.0-Komponenten abzubilden, welche sich aus <i>Kapitel 2.4.2</i> und <i>2.4.3</i> ergeben. Zusätzlich müssen die in <i>Kapitel 2.3</i> angesprochenen Assets abbildbar sein.

3.2 HERLEITUNG DES DATENMODELLS

Unter der Betrachtung vom Stand der Technik und der angestrebten Umsetzung einer I4.0-Komponente wurde ein abstraktes Beispiel genutzt. Dieses dient dazu ein Datenmodell und Datenhaltungskonzept zu entwerfen. Das Datenmodell soll hierbei die Struktur und das Verhalten einer I4.0-Komponente beschreiben und zusammen mit dem Datenhaltungskonzept die Interaktion mit einer I4.0-Komponente spezifizieren. Es wird hierfür die Produktion eines Schrankes betrachtet, welcher theoretisch produziert werden soll. Hierbei wurden die einzelnen Produktionsphasen der verschiedenen Wertschöpfungsketten durchgegangen und analysiert. Es wurde angenommen, dass zuerst ein Entwurf in Form einer CAD Skizze entstehen würde, wie sie in *Abbildung 35* dargestellt wird.

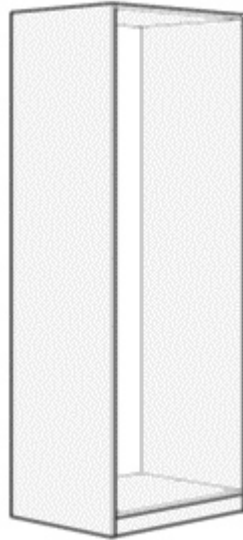


ABBILDUNG 35 PAXX-SCHRANK CAD-SKIZZE

Ausgehend von dieser CAD Zeichnung muss diese Skizze verfeinert werden. So wird die Skizze zur finalen CAD Zeichnung, welche die Zusammenhänge von 3D Objekten beschreibt, wie in *Kapitel 2.2.3.1* vorgestellt wurde. Um dieses Modell innerhalb einer I4.0-Komponente zu verwenden, muss dieses zuerst in ein abstraktes Modell überführt werden. Da jede CAD Zeichnung ein eigenes Format besitzt und somit nicht kompatibel untereinander sind, wird eine Abstraktion dieses Assets in einem einheitlichen Datenmodell benötigt. Dieses Modell muss in der Lage sein, in dieser Phase zuerst 3D Objekte mit Relationen zueinander zu verbinden und Material- sowie Produkteigenschaften zu generalisieren. Dies stellt kein Problem dar, solange sich die hierfür verwendeten Beschreibungen innerhalb der Zuständigkeit für diese Skizze befinden. Dies ist jedoch nicht immer der Fall, denn in den meisten Situationen werden z.B. Materialien einer anderen Unternehmensabteilung oder eines anderen Unternehmens verwendet. Dies bedeutet, dass eine Abbildung von solch einem Zuständigkeitsbereich existieren muss. Eine solche Zuständigkeit wird im Folgenden als Domäne bezeichnet. So werden in diesem Kleiderschrankbeispiel verschiedene Generalisierungen sowie Beschreibungen von externen Assets bezogen. Die Beschreibung des Assets könnte aussehen wie in *Abbildung 36*. Diese Abbildung repräsentiert den Kleiderschrank, welcher innerhalb der vorhergehenden CAD Skizze beschrieben wurde. Dieser wird hierfür in vier Hauptsegmente aufgeteilt, welche von der externen Asset-Beschreibung „Span Furnier“ abgeleitet wurden.

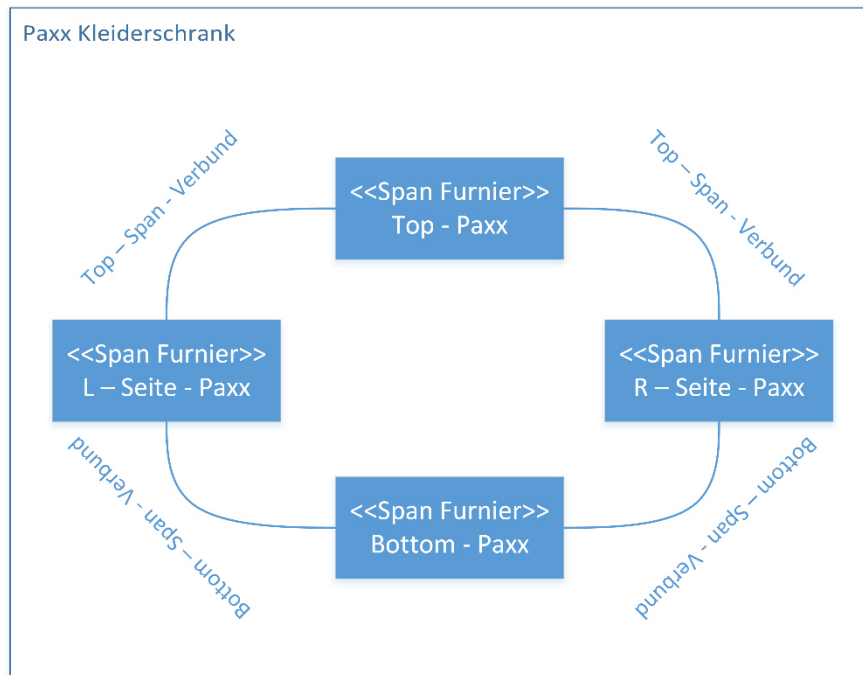


ABBILDUNG 36 PAXX - KLEIDERSCHHRANK ABSTRAKTION

Diese Ableitung wurde gewählt, um eine solche externe Abhängigkeit zu provozieren. Es könnte auch der Fall sein, dass diese innerhalb derselben Zuständigkeit sich befinden. In *Abbildung 37* wird dargestellt, wie diese externe Beschreibung aussehen könnte. Die Assets, welche in der Zuständigkeit einer Domäne sich befinden, werden als interne Assets, die außerhalb liegenden als externe Assets bezeichnet. Es wird hierbei beispielhaft das externe Asset „Span Furnier“ des internen Assets „R – Seite – Paxx“ dargestellt. Hierbei werden einzelne Domänen mit einem Rahmen um die jeweiligen Entitäten und ihrer Relationen dargestellt. Eine Relation, welche über eine solche Domäne hinausgeht, stellt die Verwendung eines externen Assets dar. Bei der genaueren Betrachtung stellt man fest, dass selten ein Element eines Assets ohne externe Abhängigkeit dargestellt werden kann. Ein solches Beispiel ist hierbei nicht unüblich, da eine solche Generalisierung bereits semantische Elemente eines Assets beschreiben kann und somit bei einer multiplen Vererbung Aufwand in der Modellierung einspart und zudem durch die Generalisierung auch eine einheitliche Bedeutung liefert. So ist innerhalb dieser Abbildung eindeutig klar, dass ein „Span Furnier“ stets ein Element vom Typ „Spanplatte“ ist, welches wiederum weitere beschreibende Attribute besitzt, wie z.B. dessen dreidimensionale Abmessung. Eine solche tiefere Beschreibung durch eine weitere Generalisierung realisiert das externe Asset „Furnier“. Dieses Asset „Furnier“ ist ausgehend von der Ursprungsdomäne, von der diese Modellierung begonnen wurde, ein externes Asset sowie von der Domäne, welche das Asset „Bottom Furnier“ beschreibt. Durch die Beschreibung dieser Ableitung von „Furnier“ auf das „Bottom Furnier“ werden hierbei die Attribute dieses Assets übertragen und auch dessen semantische Bedeutung. Die Beschreibung umfasst hierbei, dass ein „Furnier“ über eine dreidimensionale Abmessung verfügt, welche über die Assets „X-Vektor“, „Y-Vektor“ und „Z-Vektor“ beschrieben werden. In diesem Fall enthält es ebenfalls die Definition des Materials, welches durch das Element „Ressource“ abgebildet wird. Die Art vom Material wird hierbei von der Semantik der Vererbung abgeleitet, welche aus einer weiteren Domäne und von dem externen Asset „Plastig AG – PK 1127“ stammt. Im Folgenden muss geklärt werden wie auf ein solches Asset einer Domäne zugegriffen werden kann. Da nicht jede Domäne direkten Zugriff auf die anderen besitzt, wird ein an DNS angelehnter Aufbau sowie Adressierung solcher Domänen und Elemente innerhalb dieser angestrebt. Durch das gleichzeitige Anstreben einer serviceorientierten

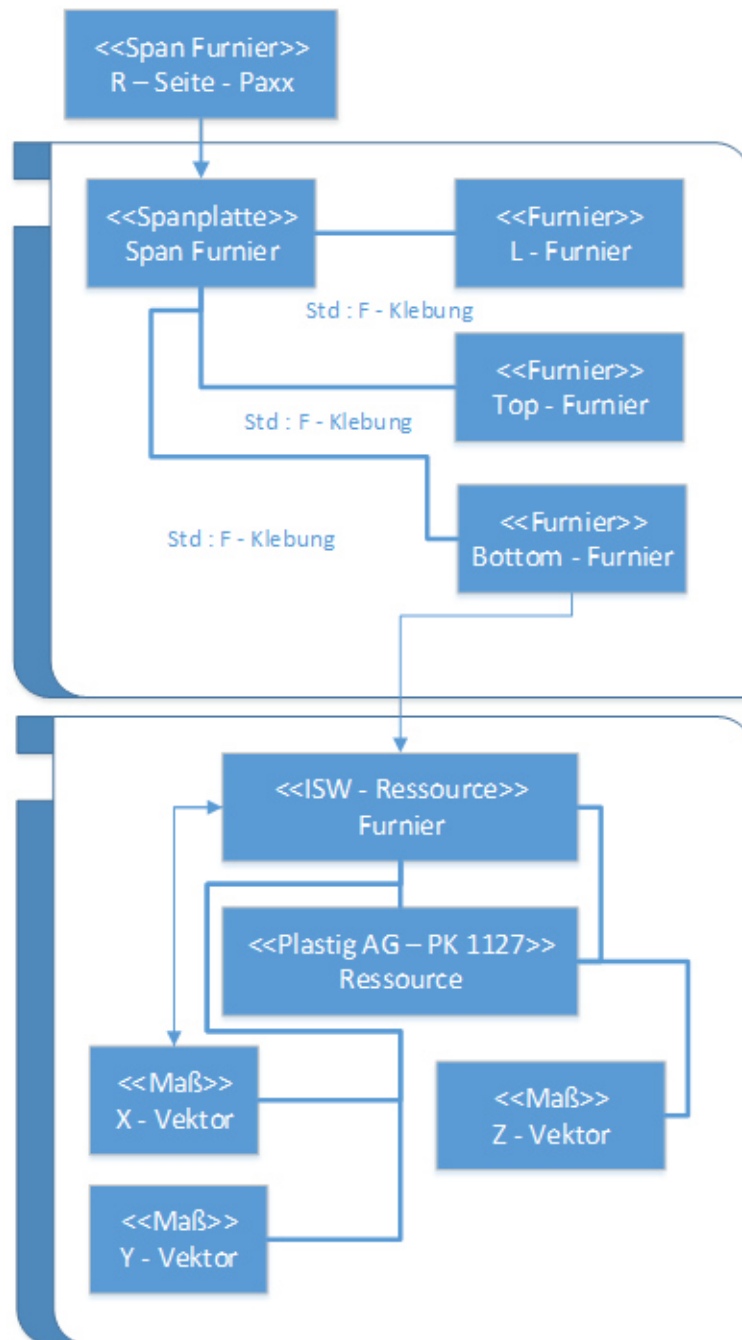


ABBILDUNG 37 EXTERNE ASSET ABHÄNGIGKEIT

Architektur, durch die Darstellung von Funktionalitäten über Service in I4.0-Komponenten, ist ebenfalls eine Registry für Service nötig, welche von einem Unternehmen bereitgestellt werden muss, um diese auffindbar zu machen. Diese beiden Eigenschaften sollen miteinander verbunden werden, so dass jede Domäne mindestens eine direkte oder innerhalb einer hierarchisch überliegenden Domäne solch eine Domänen-Registry besitzt. Da diese Arbeit die Datenhaltung von I4.0 Anwendungen beschreibt, wird hier nicht detailliert die Datenhaltung dieser Registry beschrieben. Es wird lediglich auf die aus dieser erhaltenen Daten Bezug genommen. So ist diese Registry dafür zuständig, die von einer Domäne bereitgestellten Asset-Beschreibungen zur Verfügung zu stellen. Zusätzlich werden über die Registry auch Beschreibungen von I4.0-Komponenten erreichbar gemacht. Dies ermöglicht es bei dem Erhalt von Instanzen eines Assets oder einer I4.0-Komponente auf dessen Beschreibung zu referenzieren und diese über die zuständige Domänen-Registry abzurufen. Die Funktionen dieser Domänen-Registry umfassen hierbei das Registrieren von neuen Beschreibungen eines Assets oder einer I4.0-

Komponente und das Auslesen dieser. Die Beschreibung enthält hierbei die Modellierung eines Elements und die für diese nötigen externen Beschreibungen aus anderen Domänen, um diese vollständig zu verstehen. Im Anschluss betrachten wir das Produkt, welches aus der Planung für die Herstellung eines solchen Kleiderschranks resultiert. Diese Planung beschreibt einen Prozess, welcher die Herstellung aus Grundmaterialien auf Fertigungsmaschinen nach den Vorgaben aus dem CAD Asset beschreibt. Dieser Prozess wird abstrakt in *Abbildung 38* dargestellt. Dieser besteht aus dem Ausgangsmaterial, drei Unterprozessschritten und dem Resultat, alle stellen wiederum Assets dar. Die einzelnen Prozessschritte, bestehend aus Zuschnitt, Furnierung und Bohrung, beinhalten jeweils neben der Beschreibung des Prozesses die Beschreibung der Maschine, auf der sie gefertigt werden und eine Beschreibung der Informationsquelle in Form eines Assets, welches eine CAD Datei darstellt. Ein Teil dieser Prozesserstellung wird innerhalb von Industrie 4.0 abweichen.

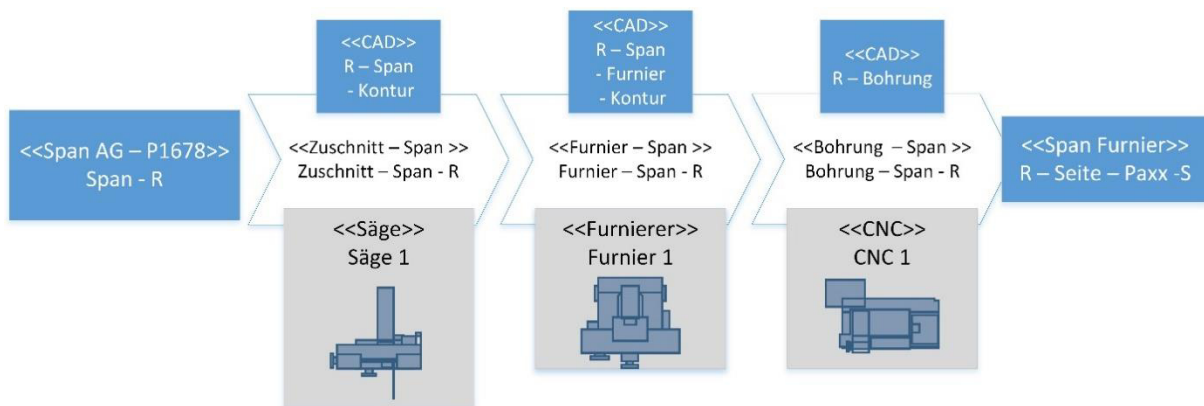


ABBILDUNG 38 TEILPROZESS-R-SEITE-PAXX

Dies resultiert daraus, dass aktuell dieser Prozess in die Instanziierung seiner Assets übergehen würde. Damit ist die Instanziierung der Produktionsanlage der nächste Schritt und in Folge dessen, die Instanziierung des Prozesses durch den Start der Produktion. Abschließend wird nach der Produktion eine Instanz des Resultates erzeugt. Dem gegenüber beginnt eine Planungsphase, wie dieser Prozess und seine beschreibenden sowie auszuführenden Assets in das I4.0-Netzwerk eines Unternehmens integriert werden. Dies kann bedeuten, dass Teile der Assets und deren Ausführungsprozess durch eine eigene Verwaltungsschale verwaltet werden oder in eine bestehende integriert werden. Der Vorteil hierbei ist, dass von den Maschinen eine zugehörige Verwaltungsschalenbeschreibung sowie deren Umsetzung Teil der Maschine als Produkt ist. So ist innerhalb des Prozesses, welcher die Produktion von dem Element „R – Seite – Paxx“ übernimmt, kein explizites Wissen über die Maschine erforderlich. Dies resultiert aus der Beschreibung des Dienstes, welcher eine I4.0-Komponente anbietet. Ein solcher Dienst ist über seinen beschriebenen Service verfügbar. Dieser wird durch ein auf Nachrichten basierendes Kommunikationsmodell realisiert, da eine lose Kopplung erwünscht ist und diese hiermit realisierbar ist. Jeder einzelne Prozessschritt kann, wie in *Abbildung 39* dargestellt, über den Service der jeweiligen I4.0-Komponente einen vereinfachten Dienst nutzen, welcher ein einheitliches Kommunikationsmodell besitzt. Hierfür muss es eine Servicebeschreibung geben, welche definiert wie ein solcher Service aufgebaut ist. Dies bedeutet, dass darin die Nachrichten für den Aufruf und dessen Resultat definiert sind. Diese Definition muss ermöglichen, „Quality of Service“ abzubilden.

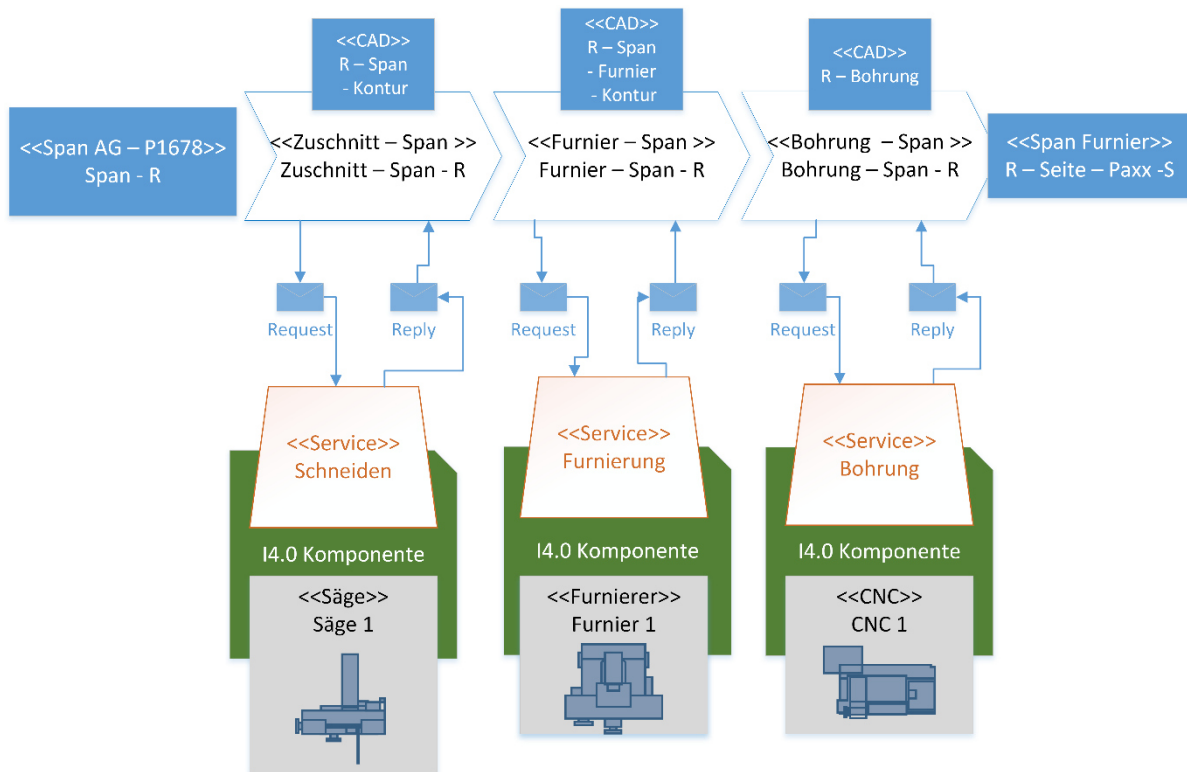


ABBILDUNG 39 TEILPROZESS MIT I4.0-KOMPONENTEN

Um eine vollständige Beschreibung zu erhalten, wie ein solcher Service die Anbindung an das Asset vollzieht, betrachten wir zunächst den Aufbau des Teilprozesses „Bohrung – Span -R“, welcher innerhalb von *Abbildung 40* präsentiert wird. Dieser bohrt vier Löcher in eine Oberfläche eines Werkstückes und besteht aus vier Unterprozessschritten, die jeweils eine Bohrung darstellen. Eine einzelne solche Bohrung entnimmt aus einem Asset, welches eine Instanz einer CAD Datei darstellt, die nötigen Informationen, um diese an den Bohrungsservice zu übertragen. Dies zeigt auf, dass hierbei das erste Mal in der Entwicklung von einer realen Instanz eines Assets gesprochen wird. Wann ein Asset eine Instanz oder eine reine Beschreibung darstellt, hängt von dessen Lebenszyklus ab. Die Tatsache, dass es sich um eine Instanz handelt, muss gekennzeichnet werden. Hierfür werden Instanzelemente zusätzlich mit einer ID versehen und einer Referenz, von welcher Beschreibung dieses Asset eine Instanz darstellt. Durch diese zusätzliche ID muss definiert werden, dass die ID eines Laufzeitobjektes an die Laufzeit seiner kontrollierenden Verwaltungsschale gebunden wird. Dies wiederum bedeutet, dass dieses Objekt nicht global eindeutig über seine ID identifizierbar ist, sondern nur innerhalb dessen Verwaltungsschale. Es ist hierdurch möglich durch die Kennzeichnung der betriebenen Verwaltungsschale und der Objekt-ID ein Objekt global zu identifizieren. Zusätzlich muss ein Prozess in der Lage sein, zu symbolisieren, welche Informationen dieser aus einem Asset bezieht und wie dies geschieht. Hierfür wurden abstrakte Interfaces eingefügt. Ein solches Interface stellt einen technologiespezifischen Zugriffspunkt dar. In diesem Fall wurde durch die blauen Verbindungen innerhalb der Abbildung signalisiert, dass sie über die ID des jeweiligen Objektes Zugriff auf seine Daten nehmen. Die grünen Verbindungen symbolisieren, welches Datenobjekt sie damit erhalten. Durch eine Pfeilrichtung kann z.B. symbolisiert werden, dass vom Ausgangspunkt auf den Endpunkt zugegriffen wird. Der Zugriff auf ein solches Interface muss spezifiziert werden, dies ist im optimalen Fall die Aufgabe eines Standardisierungsgremiums. Dieser Standard könnte dann in Form einer Asset-Beschreibung, innerhalb der Domäne dieses Gremiums, über dessen Domänen-Registry, zugänglich gemacht werden.

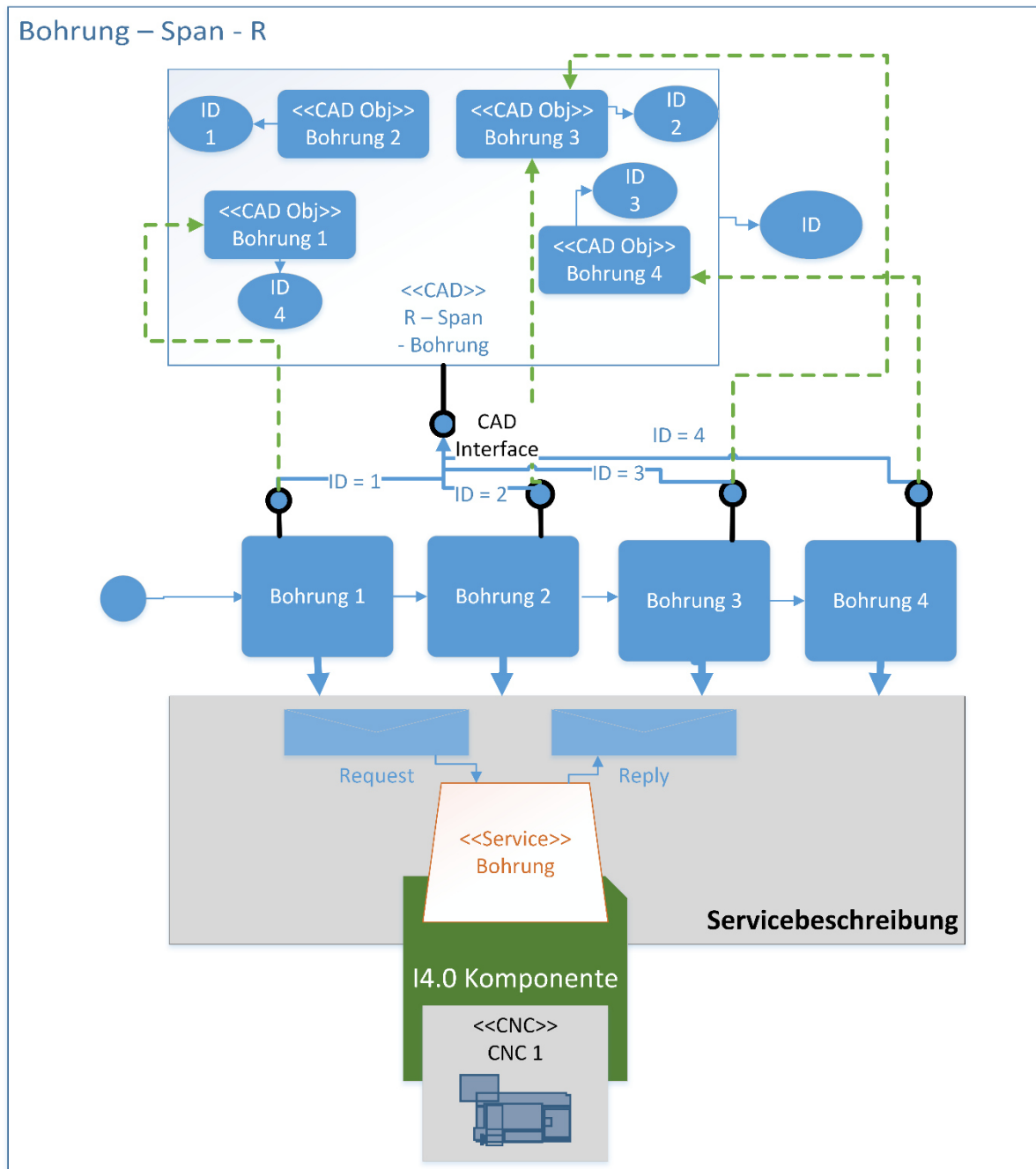


ABBILDUNG 40 SUBPROZESS BOHRUNG - SPAN - R

Dies gilt ebenfalls für die Beschreibung mancher Services, um durch die Standardisierung von diesen funktionalen Überlappungen mit mehrfachen Umsetzungen zu reduzieren. Dies setzt voraus, dass ein Service nicht direkt in seiner I4.0-Komponente definiert wird, sondern nur von diesem angeboten und auf einem Endpunkt ausgeführt wird. Die jeweiligen Bohrungsschritte verwenden die jeweils ausgelesenen Daten, um diese in das Nachrichtenformat zu überführen, welches in diesem Beispiel durch einen Bohrungsservice bereitgestellt wird. Bis zu diesem Punkt wurde betrachtet, wie Daten von Assets über deren Beschreibung generell zur Verfügung gestellt werden können und aus diesen Beschreibungen Instanzen genutzt werden, um existierende Datenobjekte hierüber darzustellen. Interfaces und ihre Verbindungselemente definieren hierdurch den Zugriff auf diese Datenobjekte. Die genutzte Servicebeschreibung stellt die Funktionalität einer I4.0-Komponente dar. Im Anschluss muss jetzt geklärt werden, wie ausgehend von einem Service innerhalb einer I4.0-Komponente die

Ausführung auf einer Asset-Instanz beschrieben wird. Hierfür visualisiert *Abbildung 41* die Beschreibung eines solchen Prozesses. Dabei wird klar, dass eine solche Interaktion einen Prozess zur Folge haben wird. Hierbei spielt es keine Rolle, ob es sich um eine reine Datentransformation oder die Ausführung eines Produktionsprozesses handelt. Es wird für die Überführung von Daten aus dem Servicekommunikationsformat und somit den Service Nachrichten ein definierter Prozess benötigt, welcher die Werte aus dem einen Format ausliest und diese in das jeweils andere überführt. In diesem Beispiel wird die Servicenachricht des Bohrungsservice genutzt, um zu illustrieren, wie eine solche Definition aussehen könnte. Durch die Servicenachricht wird der „Osaka Bohrung“-Prozess gestartet. Die Nachricht selbst enthält ein „Bohrung“-Objekt, welches die Werte für „Präzision“, „Typ“ und „Koordinaten“ enthält. Für den Ablauf des Prozesses werden diese Daten von Subprozessen verwendet, um das Asset zu steuern. Die Subprozesse selbst verwenden das von einem Asset nach außen gegebene „OPC UA Write“-Interface, um auf die „CNC 1“ Werte zu schreiben und somit den Prozess durchzuführen.

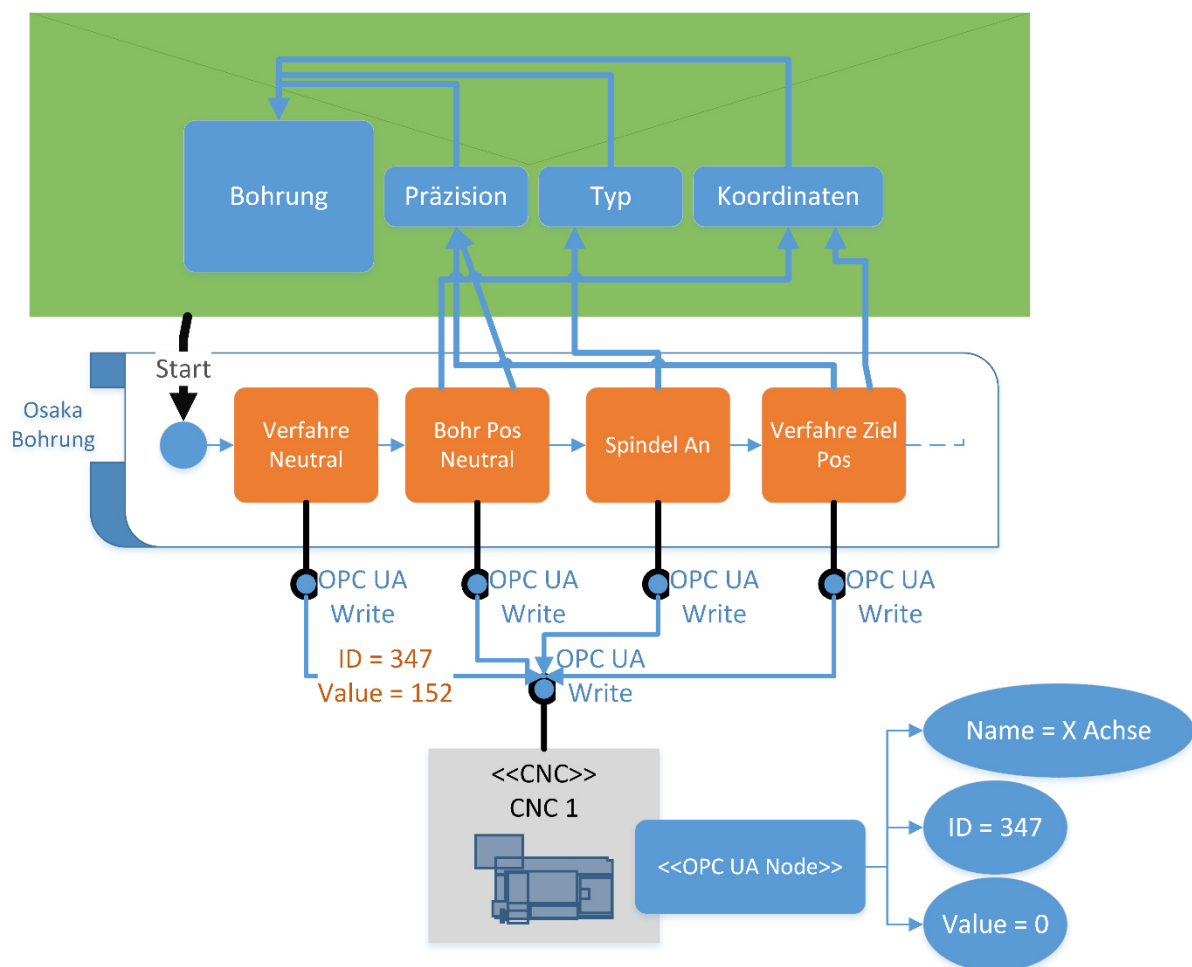


ABBILDUNG 41 BOHRUNGSPROZESS

Es wurde für eine verbesserte Übersicht lediglich die Verbindungen mit ihren Werten belegt, welche für eine abstrakte neutrale Verfahreposition nötig sind. Die weiteren Verbindungen könnten auch direkte Referenzen auf die Werte innerhalb der Nachricht aufweisen, um das direkte Mapping zu visualisieren. Es ist zudem innerhalb dieses Prozesses noch nicht nötig Werte zu definieren, welche durch den Prozess erzeugt werden, da diese noch nicht existieren. Dies liegt daran, dass ein Prozess selbst in dieser Phase einen beschreibenden Charakter besitzt und erst zu seiner Ausführung

instanziiert wird und dann die jeweiligen Verbindungen mit den dafür vorgesehenen Werten belegt. Nachdem die I4.0-Komponente beschrieben wurde, muss diese in der jeweiligen Domäne registriert werden. Dies dient dazu, um in der späteren Nutzung die Instanziierung des Modells von anderen I4.0-Komponenten oder innerhalb anderer Beschreibungen zu ermöglichen. Der vorletzte Schritt ist die Instanziierung. In dieser wird die Asset-Beschreibung innerhalb seiner zuständigen Domäne oder einer externen Domäne in Betrieb genommen. Im Gegensatz zu Assets, bei welchen es genügt diese mit einer ID zu deklarieren, ist dies bei Komponenten anders. Eine Komponente wird eindeutig durch ihre Ressourcenadresse definiert. Dies wird innerhalb dieser Arbeit eine URN Adresse darstellen. Es kann sich aber um jegliche Form der eindeutigen Adressierung handeln. Diese URN Adresse beschreibt hierbei rein die Ressource innerhalb einer Domäne. Es wird noch ein Endpunkt benötigt, auf welchem diese mit ihren Plattformservices erreichbar sein wird. Dieser Endpunkt stellt wiederum eine Ressource aus dieser Domäne dar und beinhaltet eine URL, unter welcher die I4.0-Komponente erreichbar ist. Eine I4.0-Komponente kann wie ein Service mehrere Endpunkte besitzen, von welchen die jeweilige Komponente oder der Service erreichbar ist. Der letzte Schritt besteht in der aktiven Nutzung einer I4.0-Komponente. Hierfür wird ein Service der Komponente aufgerufen, welcher hierdurch instanziiert wird. Dieser erhält eine Nachricht bezüglich seiner Servicebeschreibung. Sollte diese Nachricht nicht valide sein oder Fehler in der Ausführung auftreten, soll es möglich sein eine Fehlermeldung zu deklarieren, um auf diese reagieren zu können. Ist die Nachricht valide, wird der an diese Nachricht gebundene Prozess instanziiert. Diese Instanz ist ein neues Element innerhalb der I4.0-Komponente und wird partial zu seinem aktuellen Status aufgebaut. Hierfür werden alle Prozessteile eines Prozesses, welche über ein Interface an den Prozess oder dessen Subprozesse angebunden sind, als Instanzen vorausgesetzt. Diese Instanziierung erstellt einen Schnappschuss der Dienstauführung und der damit verbundenen Assets. Diese Abbilder können je nach Bedarf gespeichert werden. Um z.B. die Lebensdauer einer solchen Instanz darzustellen, wurden die innerhalb von Kapitel 3.3.4 präsentierten „Quality of Presentation“ eingeführt. Diese können durch die QoP Validity illustrieren, wie lange dessen Objekt mindestens existiert. Diese zusammenhängenden Informationen führen zu dem im Folgenden definierten Datenmodell, welches definiert, wie solche Services, Prozesse und Assets beschrieben werden.

3.3 DATENMODELL

Innerhalb dieses Kapitels werden die einzelnen Entitäten und ihre Relationen zueinander vorgestellt. Dieses Modell dient als Basis für eine maschinell einlesbare Umsetzung und definiert alle darin vorkommenden Objekte sowie ihrer Eigenschaften und Einschränkungen. Das gesamte Modell ist in *Abbildung 47* dargestellt. Eine solche Darstellung ist komplex, daher wird diese für eine verbesserte Übersicht im Folgenden in die fünf Sektionen I4.0 Domäne, Servicebeschreibung, Prozessbeschreibung, I4.0-Komponente und Asset unterteilt.

3.3.1 I4.0 DOMÄNE

Eine **Domäne** steht repräsentativ für eine hierarchische Strukturierung. Die oberste Domäne wird in diesem Fall eine globale Domäne sein, welche als Container für alle existierenden Domänen fungiert. Diese funktionieren so, dass eine Domäne wiederum weitere Domänen enthalten kann. Dies illustriert die „contains“-Relation in *Abbildung 42*. Das Ziel ist es die oberste globale Domäne mit Subdomänen zu füllen. Diese spiegeln die höchste Hierarchie eines Unternehmens wieder. Eine solche Subdomäne kann weitere Subdomänen enthalten, mit welchen eine hierarchische Baumstruktur aufgebaut werden kann. Diese spiegelt die Struktur eines Unternehmens wieder, welche aus weiteren Unternehmen oder Abteilungen zusammengesetzt sein kann. Eine Abteilung wäre in diesem Fall ebenfalls eine mögliche

Subdomäne eines Unternehmens. Domänen erhalten relativ zu ihrer Hierarchieebene einen eindeutigen Namen. Es soll somit möglich sein eine URN über diese entstehenden Ebenen zu erstellen, welche es erlaubt eine explizite Domäne zu referenzieren. So würde z.B. ein wissenschaftlicher Mitarbeiter mit dem Namen Max Mustermann am ISW die mögliche URN „urn:Universität Stuttgart:ISW:Max.Mustermann“ erhalten, um so auf Elemente innerhalb dessen Domäne eindeutig weiter zu referenzieren. Zusätzlich zu Subdomänen enthalten Domänen I4.0-Komponenten sowie die Definitionen von Services und Endpunkten.

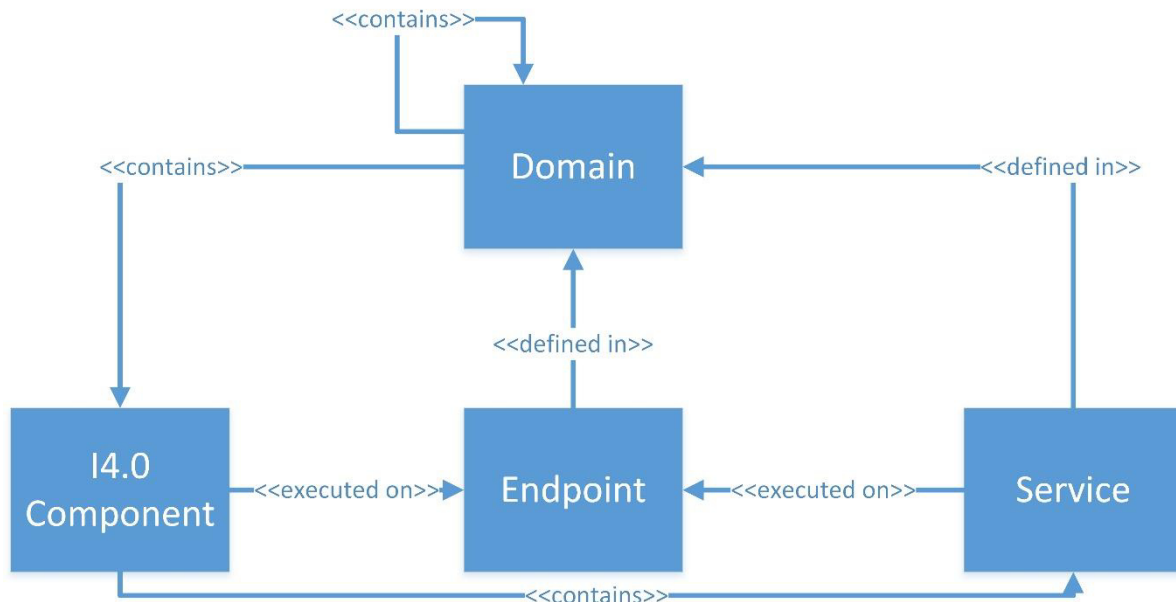


ABBILDUNG 42 I4.0-DOMÄNE DATENMODELL

I4.0-Komponenten repräsentieren das logische Element einer I4.0-Komponente aus *Kapitel 2.4.2* und wird in seiner Bedeutung innerhalb von *Kapitel 2.4.2* näher beschrieben. Der Grund für eine Definition von Services und Endpunkten innerhalb von Domänen liegt in der global eindeutigen Identifizierbarkeit dieser. Es soll hiermit erreicht werden Servicedefinitionen wieder zu verwenden und somit für Services mit der gleichen Aufgabe eine einzelne Servicedefinition zu nutzen. Services erhalten bezüglich dieses Kontextes einen relativ zu ihrer Domäne, in welcher diese definiert werden, eindeutigen Namen. Eine Service Beschreibung ist hierbei innerhalb der I4.0-Komponenten enthalten und dies bezüglich in *Kapitel 3.3.2* näher beschrieben.

Endpunkte werden aus einem ähnlichen Grund innerhalb der Domänen beschrieben, da diese logische und zugleich eindeutige Endpunkte darstellen. Dies ist nötig, da eine lose Kopplung erzielt werden muss und diese eine Referenzautonomie voraussetzt, welche wiederum logische Endpunkte benötigt. Ein solch logischer Endpunkt besitzt innerhalb dieses Modells einen relativ zu seiner Domäne eindeutigen Namen und eine Adresse in Form einer URL. Diese ist ein dynamisches Attribut, welches die aktuelle Adresse des Endpunktes beinhaltet oder den ersten Zugriffspunkt. Es handelt sich um ein dynamisches Attribut, da die URL sich jederzeit ändern kann. Der logische Endpunkt bleibt hierbei erhalten. Ein solcher Zugriffspunkt kann die übermittelten Nachrichten weiterverarbeiten oder diese an den Endpunkt weiterleiten, sollte dieser nicht bereits den physischen Endpunkt darstellen.

3.3.2 SERVICEBESCHREIBUNG

Services werden benötigt um Funktionalitäten von Assets und Komponenten mithilfe eines neutralen Datenformates zu beschreiben. Es wurde erwogen eine Web Service [28] [29] Beschreibung für diese Funktionalitäten zu verwenden. Da aber noch nicht definiert wurde, in welcher Form die Kommunikation stattfindet und zusätzlich keine Kapazitäten zur Verfügung stehen, um ein eventuell nötiges effizienteres Kommunikationsprotokoll anstelle von SOAP [29] zu erstellen, wurde ein an Web Services angelehnte Beschreibung von Services für das Modell gewählt. Der Modellabschnitt für die Beschreibung von Services befindet sich separat dargestellt in *Abbildung 43*. Diese Form erlaubt es dieses Modell auch als reine funktionale Beschreibung zu nutzen, ohne sich an eine Technologie zu binden.

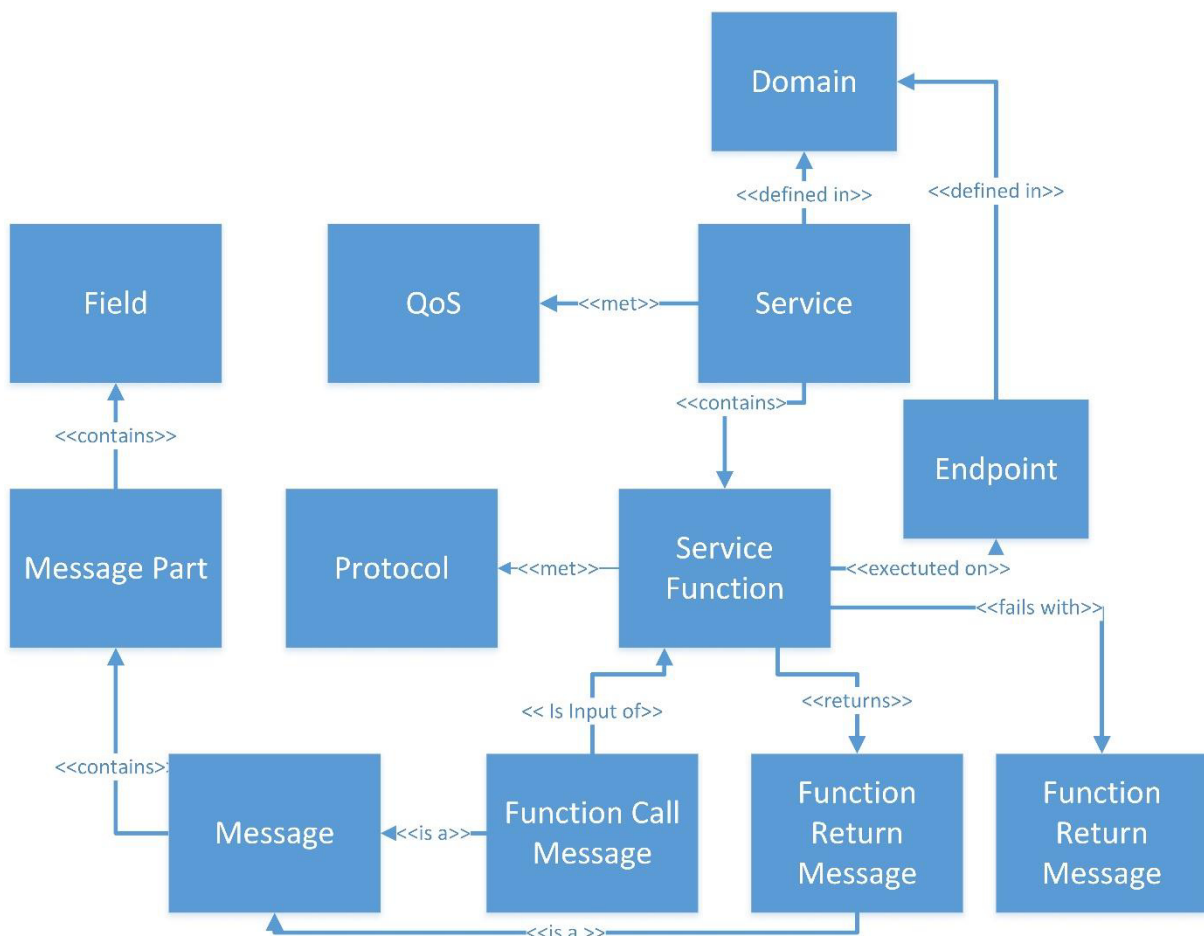


ABBILDUNG 43 SERVICEBESCHREIBUNG DATENMODELL

Ein **Service** wird innerhalb einer Domäne definiert, wie in *Kapitel 3.3.1* beschrieben. Er erfüllt **Quality of Services** [30] (QoS), welche Randeigenschaften für Geschäftsprozesse definieren. Diese Eigenschaften, welche innerhalb von *Tabelle 4* aufgelistet sind, beschreiben die Qualität eines solchen Service. Eine solche Qualität wird wiederum benötigt, um die Auswahl eines Service für eine bestimmte Situation zu ermöglichen. Als Beispiel für solch eine Situation kann ein Unternehmen dienen, welches Bahnberechnungen für eine Maschinensimulation benötigt jedoch keine Zeitlimitationen einhalten muss und hierdurch einen günstigeren oder präziseren Anbieter auswählen kann, da hierdurch auch Services genutzt werden können, welche keine Performanzangabe besitzen. Die Angabe von solchen QoS ist hierbei optional für einen Service. Werden diese angeboten, muss der Service diese auch erfüllen, sonst verliert dessen Anbieter an Vertrauen und könnte zusätzlich einen finanziellen Schaden

erleiden. Das Anbieten selbst wiederum verhindert, dass Nutzer eines Service falsche Vorstellungen von dessen Verhalten und Qualität erhalten. Neben der Erfüllung von QoS besitzt ein Service noch Servicefunktionen.

Tabelle 4 QoS Eigenschaften

Eigenschaft	Beschreibung
Erreichbarkeit	Die Erreichbarkeit ist eine Abschätzung bezüglich der Anzahl an Anfragen, die ein Dienst erhält und der Wahrscheinlichkeit, dass dieser diese Anfrage korrekt verarbeitet sowie das gewünschte Resultat erbringt. Dies bezieht sich auch darauf, dass der angefragte Dienst zu einem Zeitpunkt überhaupt auf eine Anfrage reagiert.
Integrität	Die Integrität beschreibt, wie ein Dienst die Korrektheit seiner Daten und Interaktionen gewährleistet. Dies umfasst z.B. ob Transaktionen korrekt unterstützt werden oder dass innerhalb von kritischen Interaktionen Sicherheitsmechanismen genutzt werden, welche nicht erwünschte Nebeneffekte kompensieren.
Performanz	Wie schnell ein Dienst seine Aufgabe vollzieht, beschreibt seine Performanz. Dies beinhaltet unter anderem mit welcher maximalen Latenz oder Verzögerung ein Dienst die beantragte Funktionalität liefert. Es beinhaltet ebenfalls Angaben zu seinem maximalen Durchsatz an Anfragen.
Zuverlässigkeit	Die Zuverlässigkeit beschreibt, in welchem Umfang ein Dienst in der Lage ist seine angegebene QoS zu erfüllen. Dies bezieht sich auf die benötigte Zeit innerhalb einer Zeitperiode, in welcher ein Dienst für Wartungsarbeiten nicht erreichbar ist oder diese sich auf die Erreichbarkeit auswirken.
Sicherheit	Die Sicherheit eines Service bezieht sich auf die unternommenen Maßnahmen eines Serviceanbieters, um zu gewährleisten, dass keine unautorisierte dritte Person Zugang zu den darin verarbeiteten Informationen erhält. Dies kann z.B. das Benutzen von Standards wie „Secure Kommunikation“ von Web Services sein oder auch die Verwendung von Verschlüsselungen für Daten durch RSA oder HTTPS. Es bezieht sich hierbei nicht auf das genutzte Kommunikationsprotokoll, welches von einer Servicefunktion umgesetzt wird und die Umsetzung des Transport-Layers beschreibt, sondern um die daraus resultierende Sicherheitseigenschaft.

Servicefunktionen dienen der Beschreibung von angebotenen Interaktionspunkten bezüglich eines Services. Dies können z.B. verschiedene angebotene Nachrichtenformate oder unterstützte Protokolle sein. Ein Service besitzt in seiner Beschreibung mindestens eine Servicefunktion, welche seine Funktionalität umsetzt. Servicefunktionen bieten innerhalb ihrer Definition eine Funktionsaufrufnachricht, Funktionsresultatnachricht, Funktionsfehlernachricht, das jeweilig unterstützte Kommunikationsprotokoll, ein oder mehrere Endpunkte an und besitzen einen relativ zu ihrem umliegenden Service eindeutigen Namen.

Die **Endpunkte** stellen Referenzen der innerhalb einer Domäne definierten Endpunkte dar, welche innerhalb von Kapitel 3.3.1 beschrieben wurden. Eine Servicefunktion wird an diesen Endpunkt gebunden. Dies bedeutet, dass die Initiierung einer Servicefunktion mit diesem Endpunkt assoziiert wird und daher Nachrichten an dessen definierte URL sendet. Für den Aufrufer einer solchen Servicefunktion wird diese auf dem logischen Endpunkt ausgeführt, ohne zu wissen wie dieser konkret aufgebaut ist oder wie die jeweilige Servicefunktionalität umgesetzt wird.

Nachrichten besitzen einen simplen Aufbau. Sie bestehen aus einzelnen Nachrichtenteilen, um hierdurch diese in der Größe oder bezüglich der Semantik zu trennen. Diese **Nachrichtenteile** enthalten einzelne **Felder**, welche einen relativ zum Nachrichtenteil eindeutigen *Namen*, *Typ* und *Wert* besitzen. Der Nachrichtenteil besitzt wiederum einen relativ zur Nachricht eindeutigen *Namen*. Die gesamte Nachricht besitzt relativ zur Servicefunktion eindeutigen *Namen* und relativ zur definierten Domäne eindeutigen *Typ*.

Funktionsaufrufnachrichten (FAN) stellen wie die Input-Parameter bei Programmierfunktionen die Ausgangsvariablen eines Aufrufes zur Verfügung, wenn diese benötigt werden. Sie dienen ebenso als Initiierungspunkt für Prozesse, welche innerhalb von Kapitel 3.3.3 definiert sind. Diese Nachrichten besitzen einen relativ zu ihrer Servicefunktion eindeutigen Namen und Nachrichtentyp, welcher eine Referenz bezüglich einer deklarierten Nachricht darstellt. Dies soll es ermöglichen die Struktur einer bereits definierten Nachricht wieder zu verwenden und sich auf deren Änderungen zu verpflichten. FAN besitzen ein *ReplyTo* Attribut, welches genutzt werden kann, um eventuelle Resultatnachrichten als Antwort zu erhalten. Der darin übergebene Endpunkt muss nicht mit dem des Aufrufers übereinstimmen. Es handelt sich hierbei um FRN oder FFN der beiden folgenden Beschreibungen.

Funktionsresultatnachrichten (FRN) dienen der Übermittlung des Resultates einer Dienstauführung. Diese besitzt einen relativ zu ihrer umliegenden Servicefunktion eindeutigen *Namen* und eine Referenz zu ihrem *Typ*, welcher dessen Struktur beschreibt. Zusätzlich hierzu wird ein *ResultType* angegeben. Dieser spiegelt drei Formen von Resultaten wieder. Es handelt sich hierbei um ein einzelnes direktes Resultat, Events einer Subscription und Request Resultate. Ein Resultat besteht aus einer einzelnen Nachricht. Events dienen der Beschreibung von Subscriptions, welche nicht aus einer Nachricht bestehen, sondern mehrere Nachrichten eines Typs besitzen. Diese können solange an einen Endpunkt versendet werden bis diese erneut abbestellt werden. Request Resultate sind Nachrichten, welche die Information enthalten, dass ein Objekt abgerufen werden kann. Hierfür enthält diese Nachricht die URN dieses Objektes und dessen Endpunkt an welchem es für einen Abruf bereitsteht. Dies ermöglicht große Objekte auf Anfrage bereit zu stellen und innerhalb dieses Objektes Information über den aktuellen Status über dessen Bereitstellung zu liefern.

Funktionsfehlernachrichten (FFN) sind Nachrichten, die vordefinierte Fehlerstadien eines Dienstes repräsentieren und Informationen übermitteln, welche den Fehler oder das inkorrekte Verhalten des Dienstes näher beschreiben. FFN folgen dem *ResultType* von FRN, um eine einheitliche Nutzung zu erlauben. Sie treten somit anstelle der FRN auf. Diese Kategorie der Nachrichten besitzt einen relativ zur umliegenden Servicefunktion eindeutigen *Namen*.

3.3.3 PROZESSBESCHREIBUNG

Prozesse stellen das Verbindungsglied zwischen dem Serviceaufruf und dessen Resultat dar. Sie sind Teil der I4.0-Komponenten und der darin liegenden Assets, um ihre Funktionalitäten und Abhängigkeiten zu beschreiben, wie innerhalb von *Kapitel 3.3.4* dargestellt. Die in *Abbildung 44* illustrierten Entitäten und ihre Relationen zueinander beschreiben die Elemente, welche nötig sind, um solche Prozesse darzustellen. Die Prozessbeschreibung ist an BPMN und BPEL angelehnt. Ein Prozess wird durch die Verknüpfung von Aktivitäten, Gates, Events, Nachrichten und Datenobjekten durch Flows verbunden.

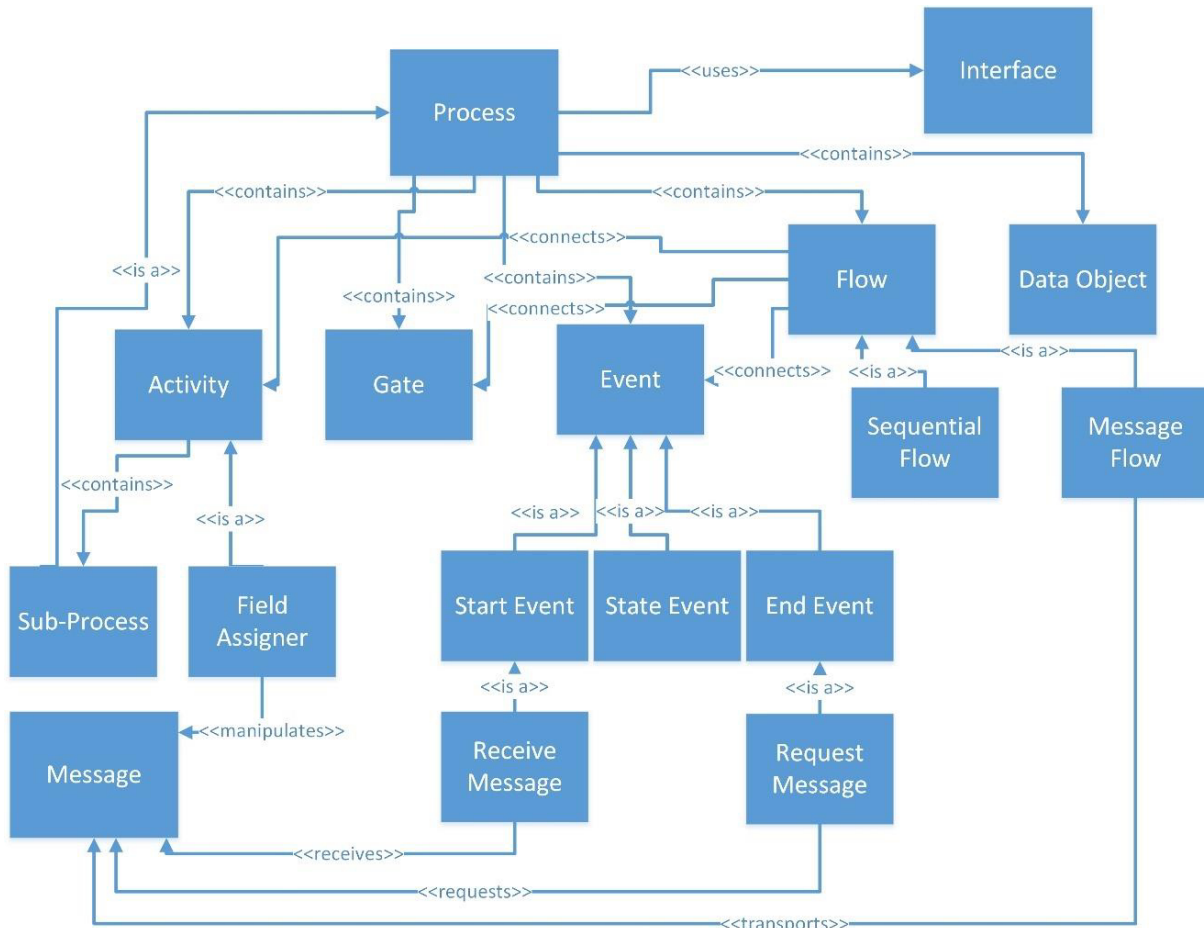


ABBILDUNG 44 PROZESSBESCHREIBUNG DATENMODELL

Events treten in den drei verschiedenen Grundsemantiken **Startevent**, **Zustandsevent** und **finale Event** auf. Startevents sind der Eintrittspunkt in jeden Prozess oder eines verschachtelten Subprozesses. Von einem solchen Event werden mithilfe von Flows weitere Aktivitäten oder Gates ausgelöst. Es gilt hierbei das ein Event immer nur einen ausgehenden Flow besitzen kann, da ein Event selbst keine komplexe Logik besitzt und zuständig für die Signalisierung eines Zustandswechsels des überliegenden Prozesses ist. Finale Events dienen als Endpunkte eines Prozesses. Sie besitzen keine eigenen ausgehenden Flows mehr. Es ist aber möglich, von einem anderen Prozess aus, einen Flow auf ein solch finale Event zu legen, um so einen weiteren Prozess am Ende eines vorhergehenden Prozesses oder eines Subprozesses ausführen zu können. Zustandsevents entsprechen Zwischenevents, welche das Erreichen eines ausschlaggebenden Punktes innerhalb einer Prozessausführung darstellen. Das kann z.B. das Abschließen einer Transaktion darstellen, welche wieder kompensiert werden muss, sollte der gesamte Prozess abgebrochen werden. Besondere Events stellen das „Nachrichten empfangen“-Event und das „Nachricht anfragen“-Event dar. Diese sind dafür zuständig bestimmte Nachrichten zu empfangen oder anzufragen, um so Informationen in den Prozess zu bringen oder als Resultat als eine Nachricht zu versenden.

Gates sind einzelne Entscheidungspunkte eines Prozessflusses. Es existieren die vier Gates „paralleles-Gate“ (pGate), „oder-Gate“ (oGate), „konditionales Gate“ (kGate) und „Synchronisation-Gate“ (sGate). Das sGate dient der Synchronisation einer unbekanntem Menge an parallel ablaufenden Prozessflüssen, um diese auf einen Punkt in der Zeit zu synchronisieren. Ein „oGate“ stellt eine „entweder oder“-Entscheidung dar und beinhaltet eine Bedingung. Ist diese zum aktuellen Zeitpunkt der Ausführung erfüllt, wird der dafür vorgesehene Pfad ausgeführt, andernfalls folgt der ungültige Pfad. Dies setzt voraus, dass ein „oGate“ immer zwei folgende Flows besitzt, welche diese beiden Pfade repräsentieren. Das „pGate“ stellt den Beginn einer asynchronen Ausführung dar und ermöglicht es nicht sequentiell ablaufende Prozesse abzubilden. Die Form des „kGate“ ist die Repräsentation einer komplexen Logik, in welcher mehrere Pfade existieren und hierdurch auch mehrere Bedingungen für einen Pfad. Im Gegensatz zu „oGates“ ist es möglich mehrere Pfade gleichzeitig auszuführen, sollten multiple Pfadbedingungen erfüllt sein.

Ein **Flow** besitzt eine Richtung. Diese definiert die Flussrichtung des Prozesses. Es existieren die zwei Arten sequentieller Flow und Nachrichten-Flow. Der sequentielle Flow definiert das nächste folgende Element, hierbei kann es sich um eine Aktivität, ein Gate oder Event handeln. Der Nachrichten-Flow besitzt eine ähnliche Aufgabe, ist aber darauf limitiert, dass er nur nachrichtenbasierende Elemente miteinander verbinden kann. Das Ziel von Nachrichten-Flows ist es, Prozesse zu verbinden und Informationen zwischen diesen auszutauschen. Zusätzlich sind diese Flows immer mit einer Nachricht assoziiert. Diese Nachricht wird zu einem Service eines Assets oder einer Komponente gehören, welcher mit dieser Nachricht angefragt wird. Dieser repräsentiert wiederum einen Prozess, welcher als Resultat eine Nachricht enthält. Die resultierende Nachricht wird von einem „Nachricht erhalten“-Event konsumiert, welches wiederum ein Startevent für einen Prozess darstellt.

Aktivitäten sind eine abstrakte Darstellung von realen Prozessen oder Interaktionen eines Systems. Sie können atomare Aktionen repräsentieren, welche nicht in weitere Teilprozesse untergliedert werden können oder einen weiteren gekapselten Prozess darstellen. Dieser Prozess besteht aus denselben Bestandteilen wie der darüber liegende Prozess. Es existiert für diese Subprozesse nur eine zuzügliche Regel. Jeder Flow der von einer Aktivität startet oder in diese endet, muss einem dazugehörigen Event zugewiesen werden, wenn multiple Start- oder finale Events möglich sind. Dies kann entfallen, wenn nur eines der jeweiligen Events vorliegt und somit eine Zuweisung eindeutig macht. Eine Prozessaktivität stellt der **FieldAssigner** dar. Dieser ist eine für den Transfer von Wertebelegungen zwischen Nachrichten und Datenobjekten relevante Aktivität. Sie kann z.B. dafür verwendet werden, um Rückgabewerte eines vorausgegangenen Serviceaufrufs zu nutzen und die darin enthaltenen Werte für einen neuen Serviceaufruf an ein Asset oder eine I4.0-Komponente weiterzuleiten.

Datenobjekte stellen eine abstrakte Form eines Datenspeichers dar. Ein Datenobjekt ist wie eine Nachricht kein Event, Flow, Gate oder eine Aktivität. Daher können Datenobjekte nur mit einer Aktivität oder Nachrichten Flow assoziiert werden. Diese Assoziation signalisiert, dass hierbei Daten aus dem Datenobjekt gelesen oder in dieses geschrieben werden. Datenobjekte können in diesem Kontext z.B. Datenbanken oder auch für diesen Prozess nötige temporäre Datenspeicher sein. Es ist nicht vorgesehen, dass diese Informationen aus Dateien geladen werden, da diese innerhalb des I4.0 Kontextes als Assets gehandhabt werden und somit dessen Informationen über deren Service oder Interface abgerufen würden. Dies wird in Kapitel 3.3.4 detaillierter besprochen.

3.3.4 ASSET

Ein **Asset** stellt innerhalb dieses Modells die Abbildung eines I4.0 Assets dar. Es besitzt drei Arten von Elementen, welche dafür zuständig sind aus einer assetspezifischen Beschreibung die Informationen zu normalisieren und nach außen zu tragen, die Herkunft der Quelldaten zu beschreiben und das Asset an sich zu definieren. Die für ein Asset relevanten Relationen und Elemente sind in *Abbildung 45* dargestellt.

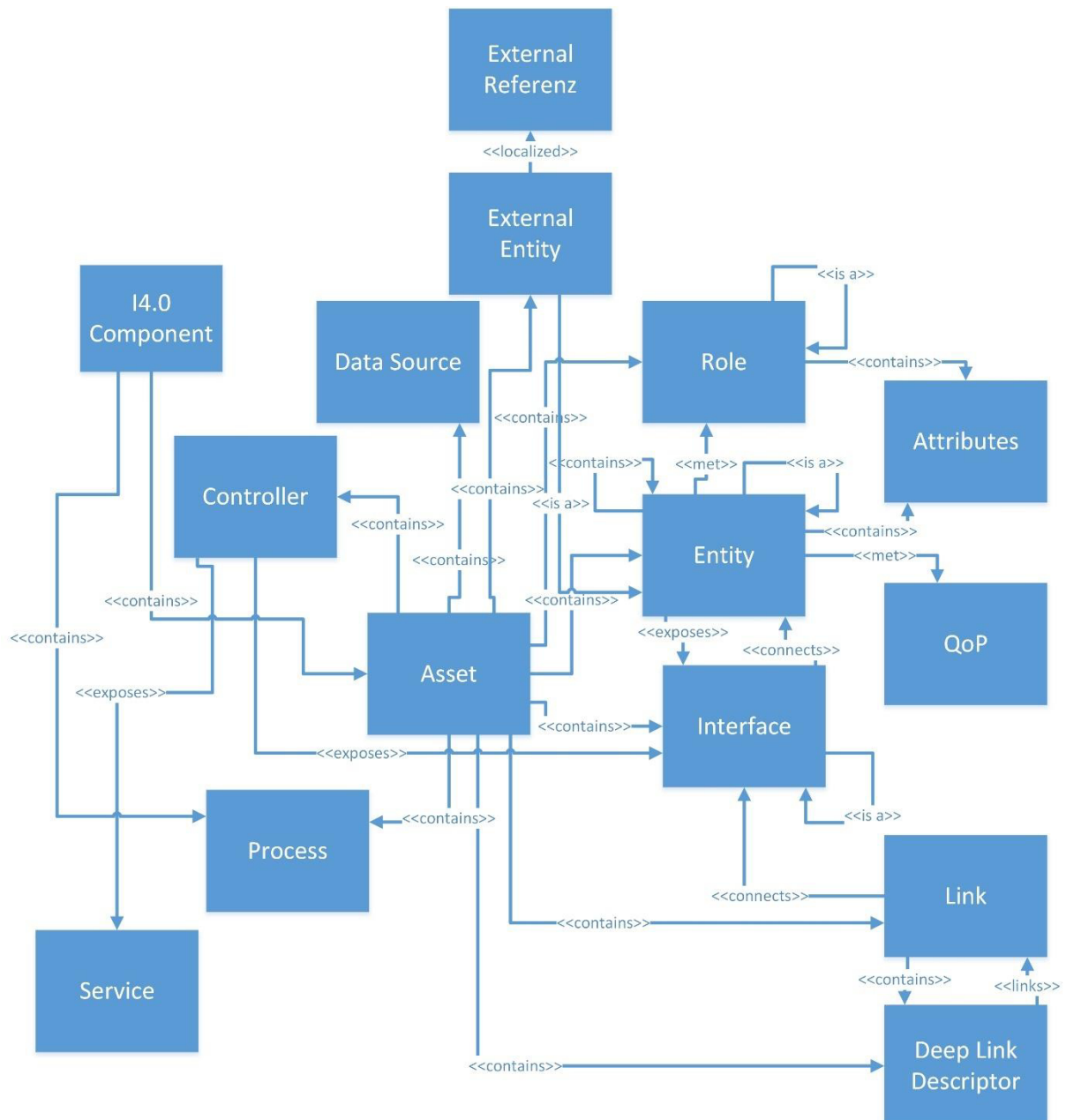


ABBILDUNG 45 I4.0 ASSET DATENMODELL

Ein **Datenquellenelement** dient der Lokalisierung der Daten, welche ein Asset beschreiben. Das Datenquellenelement besitzt innerhalb seines Assets einen eindeutigen *Namen* und einen *Accesspoint* zu einer *Ressource*. Ein solcher Accesspoint enthält eine URN Referenz auf die Ressource, Attribute welche Informationen bezüglich des Zugriffes auf diese Ressource kapseln und den dazugehörigen *Quellentypen*. Dieser Quelltyp spezifiziert was für eine Art von Quelle genutzt wird und repräsentiert, z.B., dass es sich um ein CMS handelt, an welchem die Daten angefragt werden können. Zusätzlich

enthält es weitere Attribute und einen *Typ*, welcher beschreibt was für eine Art von Datenelement genutzt wird. Diese Attribute dienen spezifisch zur Datenquelle einer näheren Lokalisierung der asset-beschreibenden Elemente. Ein Asset kann hierbei mehrerer solcher Quellen besitzen, sollten die Informationen über dieses auf mehrere Informationspunkte verteilt sein. Die angegebene URN referenziert einen Endpunkt innerhalb einer Domäne, welcher die angegebene Ressource verwaltet. Die Daten, welche innerhalb der Datenquelle beschrieben sind, werden innerhalb eines Assets über Entitäten abgebildet.

Diese **Entitäten** bilden ein abstrahiertes Objekt ab, welches eine digitale Repräsentation einer Eigenschaft oder eines Objektes widerspiegelt. Hierfür unterstützen Entitäten eine Form der Vererbung, was beinhaltet, dass diese eine Referenz auf die zu erbende Entität erhält, all seine Attribute, Entitäten, Rollen und Interfaces übernimmt und deren eigene Beschreibung erfüllt. Diese Beschreibung besteht aus denselben Elementen, welche durch die Vererbung weitergegeben werden, mit der Ausnahme, dass diese keine Referenz auf die vererbte Entität erhalten. Wird ein reales Objekt über eine solche Entität dargestellt, erhält es „**Quality of Presentation**“-Eigenschaften. Diese beschreiben die Qualität der Darstellung eines Objektes. Es werden die in *Tabelle 5* aufgelisteten QoP unterstützt. Das Ziel ist es mit diesen QoP, ähnlich wie mit den QoS, Aufschluss über das Verhalten eines Objektes zu geben und Garantien bezüglich dessen Nutzung anzubieten. Ein Asset beinhaltet ebenfalls Rollen, welche von einer Entität erfüllt werden kann.

TABELLE 5 QUALITY OF PRESENTATION EIGENSCHAFTEN

Eigenschaft	Beschreibung
Validität	Es wird hierbei beschrieben, in welchem Zeitaspekt eine dargestellte Entität mit seiner aktuellen Belegung als gültig angesehen werden kann. Dies kann z.B. das Intervall einer zyklisch aktualisierten OPC UA Variablenrepräsentation widerspiegeln, so würde dessen Wertebelegung bis zum nächsten Zeitpunkt der Aktualisierung valide sein.
Latenz	Die Latenz gibt eine Verzögerung an, in welcher das innerhalb des Modells dargestellte Objekt bezüglich der Zeit von dem Original abweicht. Dies ist besonders relevant, wenn Assets Entitäten anderer Assets verwenden, da diese keinen direkten Zugriff auf diese Objekte besitzen. In diesem Fall wird es immer eine Latenz durch die Repräsentation geben.
Detaillevel	Die Darstellung eines Objektes kann aus mehreren Gründen nicht der vollständigen Beschreibung entsprechen. Es kann der Komplexität einer Entität geschuldet sein, sollte diese umfangreich ausfallen oder aus Sicherheitsgründen nicht gestattet sein ein Element darzustellen. Dies findet z.B. Verwendung, wenn ein Asset externe Entitäten verwendet, jedoch keinen Zugriff auf deren verbundenen Entitäten und Beschreibung besitzt. Hierdurch können diese Entitäten als nicht vollständig oder verschleiert gekennzeichnet werden.

Externe Entitäten stellen eine spezielle Form von Entitäten dar und repräsentieren den Zugriff auf Daten oder Funktionalitäten eines anderen Assets, welches sich innerhalb derselben I4.0-Komponente oder externen I4.0-Komponenten befindet. Sie werden genutzt, um die Verbindung zweier physikalisch verbundener Assets zu repräsentieren. Dies könnte z.B. die Verwendung einzelner OPC UA Variablenwerte innerhalb einer weiteren Asset-Instanz darstellen. Da Entitäten nur über ihre Interfaces mit weiteren Entitäten verbunden werden können, ist der Zugriff auf diese klar definiert. Eine externe Entität wird wie eine Entität eines Assets behandelt, erhält jedoch eine QoP-Latenz

Eigenschaft. Dies resultiert daraus, dass die Informationen nicht ohne eine Verzögerung zwischen Assets übertragen werden können.

Die **externe Referenz** beschreibt die Lokalisierung einer Entität. Sie beinhaltet Informationen über die zuständige Verwaltungsschale, das umliegende Asset und die repräsentierte Entität. Wird eine externe Entität verwendet, erhält diese eine externe Referenz.

Rollen beinhalten ähnlich wie Entitäten Attribute und bieten Interfaces nach außen an. Sie können aber keine direkte Instanz bilden und dienen der reinen semantischen sowie strukturellen Beschreibung von Entitäten. Eine Rolle erfüllt das Ziel einer Abstraktion eines Systems und dessen Akteuren. Es ist hierbei nicht von Bedeutung, ob es sich um eine logische oder reale Abstraktion handelt. So kann z.B. eine Rolle die Abstraktion eines Fließbandes sein, welches zwei Interfaces besitzt, um eine Entität aufzunehmen und diese nach dem Transport wieder abzugeben. Diese Rolle kann durch zusätzliche Attribute semantisch genauer definiert werden. Im Anschluss wird diese Rolle einer Entität zugewiesen. Der Vorteil hierbei ist, dass ein Teil der Bedeutung einer Entität durch ihre Rolle definiert wurde. So ist es ebenfalls möglich Platzhalter-Entitäten in eine Struktur einzusetzen, welche über eine beliebige andere Entität ersetzt werden kann. Diese muss dieselben Rollen erfüllen wie die Platzhalter-Entität. Für die Erweiterung einer semantischen Bedeutung unterstützen Rollen ebenfalls Vererbung. Dies resultiert in einer „is a“ Relation bezüglich der Rolle und ihrer Vaterrolle. Es werden hierbei alle Attribute und Interfaces übernommen und anhand einer Referenz auf die Vaterrolle, die Vererbung repräsentiert. Vererbte Rollen dienen der genaueren Spezifizierung von Eigenschaften einer Rolle. Es ist ihnen nicht gestattet Interfaces oder Attribute zu entfernen, welche durch die Vererbung erlangt wurden. Dies gewährleistet, dass eine Rolle die semantische Bedeutung der ererbten Rolle übernimmt und diese erweitert.

Ein **Link** besteht zwischen zwei Interfaces von Entitäten oder Rollen. Es ist nicht zulässig, dass ein Interface von einer Rolle mit dem Interface einer Entität verbunden ist, da es kein reales Instanz-Objekt von einer reinen Rolle geben kann. Links bilden den Informationsfluss zwischen Entitäten über ihre Interfaces ab. Besteht ein Link zwischen zwei Rollen-Interfaces bedeutet dies, dass wenn eine Entität eine Rolle erfüllt, muss das Interface, welches die Entität durch die Vererbung der Rolle erhalten hat, mit dem Interface einer Entität verbunden werden, welches dieses durch Vererbung von der zweiten Rolle erhalten hat. Ein solcher Link zwischen zwei Rollen beschreibt somit eine rein semantische Eigenschaft, welche von Instanz-Entitäten umgesetzt wird. In Situationen, in denen ein Interface aus mehreren Interaktionen zwischen Entitäten realisiert wird, enthalten diese auch multiple Links untereinander. Damit die Information nicht verloren geht, dass ein Link mehrere Links zu seiner Ausführung benötigt, ist es möglich **DeepLinks** an einen Link hinzuzufügen. Diese beinhalten eine Referenz an alle Links, welche eine Relation zu diesem Link besitzen.

Das letzte für die Beschreibung von assetsspezifischen Elementen ist der **Controller**. Er beschreibt die für I4.0-Komponenten spezifischen Dienste, welche ein Asset an Funktionalität nach außen trägt. Es ist hierbei ähnlich zu den Diensten, die von I4.0-Komponenten selbst angeboten werden und dienen der Normalisierung von der spezifischen Asset-Beschreibung in eine I4.0 Servicebeschreibung. Die Absicht dahinter ist über eine Servicebeschreibung eine bestimmte Funktionalität einheitlich zu beschreiben und diese Beschreibung wiederzuverwerten. Zusätzlich ist es möglich assettspezifische Interfaces über den Controller nach außen zu geben, um diese für andere Assets oder assettspezifische Prozesse zugänglich zu machen. Dies ermöglicht es im Gegenzug Assets für eine bestimmte Funktionalität austauschbar zu machen, da hierdurch definierte Schnittstellen entstehen. Die Servicebeschreibung

eines Controllers entspricht der in *Kapitel 3.3.2* erläuterten Struktur. Der Controller selbst beinhaltet neben den Services, die er zur Verfügung stellt, einen Prozess, welcher für sein Asset spezifische Aktivitäten enthält. Dieser Prozess ist dafür zuständig den Ablauf zu definieren, wie das jeweilige Asset die Servicefunktionalität umsetzt und die Resultate bereitstellt. Die Möglichkeit Services ebenfalls in Assets zur Modellierung zur Verfügung zu stellen, resultiert aus der Tatsache, dass Assets wie z.B. ein OPC UA Server bereits ein nachrichtenbasiertes Interface besitzen. So würde diese Information verloren gehen, wenn über den Controller nur Interfaces nach außen freigegeben werden. Hierdurch ist es möglich Services von Assets direkt innerhalb des neutralen Datenformates zu nutzen, ohne weitere Eigenschaften für Prozessaktivitäten zu spezifizieren.

3.3.5 I4.0 KOMPONENTE

Eine I4.0-Komponente, welche innerhalb von *Abbildung 46* dargestellt ist, besteht aus den Elementen Services, Prozesse und Assets. Eine Komponente selbst besitzt bis auf die definierten Prozesse keine weitere Beschreibung. Sie müsste in einem Asset beschrieben werden und mithilfe eines Service oder Interface von einem Asset zur Verfügung gestellt werden. Die Aufgabe einer I4.0-Komponente besteht darin eventuell physisch verteilte Services logisch über eine I4.0-Komponente darzustellen. Es werden hierfür die innerhalb der Domäne definierten Services mit Endpunkten bereitgestellt und deren Aufruf an Prozesse gekoppelt. Diese Prozesse können die simple Verkettung von Asset-Diensten sein oder auch die Realisierung von einer Delegation anderer I4.0-Komponentenservices, welche von einem externen Netzwerk aus nicht erreichbar sein sollen. Dieses kann mit einer Überprüfung von Zugriffsrechten oder der Anreicherung von Daten versehen werden, welche aus anderen Diensten von Assets oder Verwaltungsschalen hervorgehen. Alle Elemente, die innerhalb einer I4.0-Komponente enthalten sind, dienen der Beschreibung von Elementen und ihrer Relationen zueinander. Wird auf eine I4.0-Komponente durch seine Anwendungsdienste zugegriffen, erfolgt die Instanziierung der Beschreibung bezüglich des ausgeführten Dienstes und seines verbundenen Prozesses. Eine I4.0 Komponente ist über mindestens einen Endpunkt erreichbar, der die nötigen Plattformservices bereitstellt, welche in *Kapitel 2.4.2* beschrieben wurden. Diese dienen der allgemeinen Nutzung von einer I4.0-Komponenten, um Elemente aus dessen digitalen Darstellung auszulesen oder zu manipulieren. Es ist hierbei wichtig zu berücksichtigen, dass die Autorisierung nicht definiert wurde. Es wird davon ausgegangen, dass der Servicebus dies vollzieht, sollte eine Anfrage nicht valide sein.

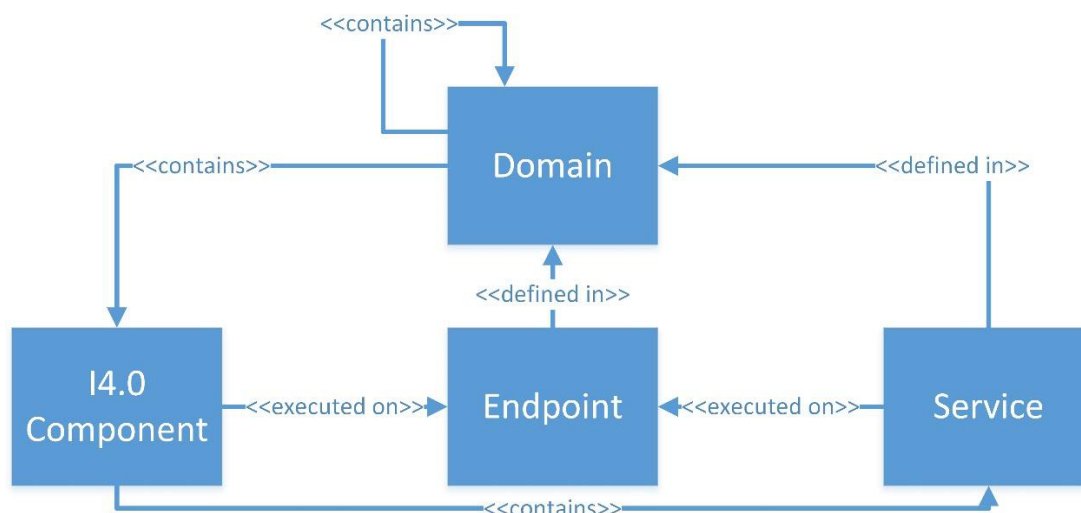


ABBILDUNG 46 I4.0-KOMPONENTE DATENMODELL

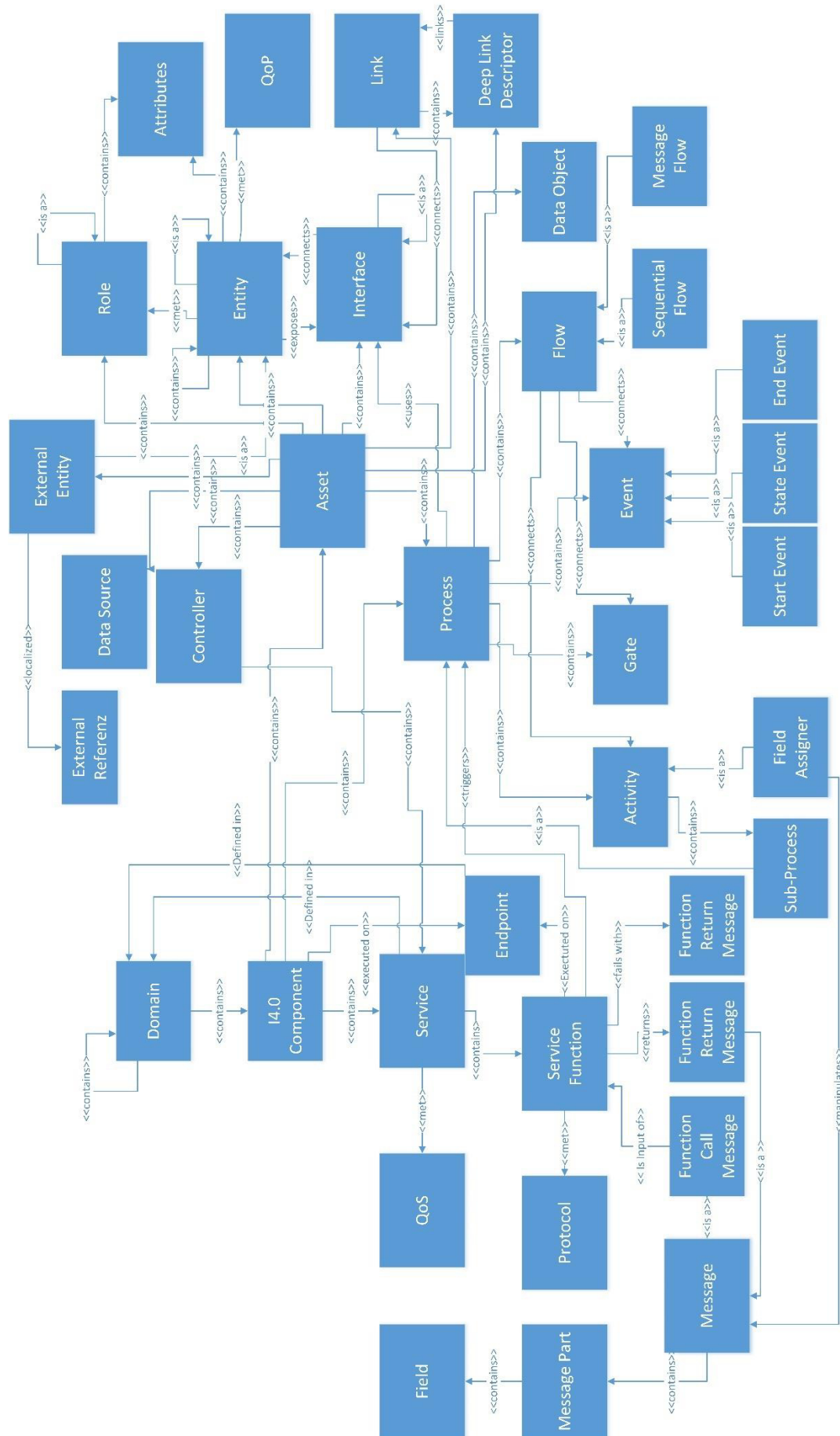


ABBILDUNG 47 GESAMTES DATENMODELL

4 UMSETZUNG

Dieses Kapitel befasst sich mit der Überführung des abstrakten Datenmodells in eine maschinenlesbare Form. Für diese wurde AutomationML als Datenformat gewählt. Diese Entscheidung wird zunächst im *Kapitel 4.1* begründet. Im Folgenden wird innerhalb von *Kapitel 4.2* definiert wie die Daten mittels AutomationML verwaltet werden müssen. Das anschließende *Kapitel 4.3* stellt das Datenmodell in AutomationML dar. Abschließend wird in *Kapitel 4.4* anhand eines Beispiels das Modell validiert und in *Kapitel 4.5* die Erfüllung der Anforderungen an das Datenmodell präsentiert.

4.1 WAHL DES FORMATES

Für eine beispielhafte Umsetzung wurde AutomationML als Format gewählt, da es ein auf XML basierendes Format darstellt. Dies bedeutet, dass hierdurch bereits eine Plattformunabhängigkeit in der Darstellung erreicht wird. Zusätzlich ist die Aufgabe von AutomationML Assets, entlang der Wertschöpfungskette übergreifend, zu definieren und zu verwenden. Hierfür werden zusätzlich standardisierte Rollen- und Interfacebibliotheken bereitgestellt, welche somit bereits eine Grundlage für die Beschreibung von Assets bieten. Da es sich hierbei um ein XML-Format handelt, kann dieses auch bei Bedarf erweitert werden und weitere XML-Definitionen nutzen. AutomationML ermöglicht benötigte Funktionalitäten wie hierarchische Strukturen, Vererbung, Darstellung von Rollen, Interfaces von Entitäten, die Verbindung von Interfaces und eine definierte Darstellung von der Beschreibung eines Elements sowie dessen Instanz-Beschreibung. Dies wiederum deckt einen Großteil der Darstellungsfunktionen ab, welche für die Darstellung des Datenmodells aus *Kapitel 3.3* benötigt werden. Es wurden zusätzlich die Entwicklung eines proprietären Formats in YAML, XML oder JSON erwogen, jedoch aufgrund des Aufwandes für die Standardisierung eines wohldefinierten Formates verworfen. Zudem würden hierbei jegliche Asset-Beschreibung eine erneute Standardisierung benötigen und zusätzliche Validierung, ob dieses Metamodell zur Asset-Modellierung ausreichend ist. Dies übersteigt die Kapazitäten dieser Arbeit und würde nicht zu der Validierung des Modells beitragen. Daher wird für die Realisierung AutomationML verwendet. Wie die Daten aus AutomationML verwaltet werden wird innerhalb von *Kapitel 4.2* präsentiert und das Mapping von dem abstrakten Datenmodell durch AutomationML in *Kapitel 4.3*.

4.2 DATENHALTUNG MIT AUTOMATIONML

In diesem Kapitel wird die Datenhaltung mit der Verwendung von AutomationML besprochen. Es wird hierbei geklärt, wie die Beschreibung von Assets, I4.0-Komponenten und Services abläuft. Dabei werden erläutert, wie Segmente innerhalb der drei AutomationML Bibliotheksstrukturen verwaltet werden und die Nutzung der I4.0-Komponenten Plattformservices, um mit einer betriebenen I4.0-Komponente zu interagieren. Die abstrakte Beschreibung von I4.0-Komponenten, Assets und Services befindet sich innerhalb der zuständigen Domänen-Registry. Diese lässt ebenfalls einsehen, welche Services und I4.0-Komponenten auf Endpunkten dieser Domäne erreichbar sind. Diese Beschreibung befindet sich innerhalb einer AutomationML-Datei, welche über die Domänen-Registry zugänglich gemacht wird. Diese besitzt den in *Abbildung 48* dargestellten Aufbau. Es wurde hierbei auf die theoretische Domänen-Registry des „ISW“ zugegriffen, welches eine Unterdomäne von „Universität“ darstellt. Das Dokument wurde unter dieser Domäne als „Domain-Registry-Example“ bezeichnet. Es beinhaltet vier Systemklassenbibliotheken, eine Rollenklassenbibliothek und eine Interfaceklassenbibliothek. Die Benennung dieser erfolgt simultan zu ihrer URN Adresse, somit beginnen alle Bibliotheken mit der URN „Universität:ISW“. Die jeweils darunterliegenden Elemente werden über ihren Namen und ihrer hierarchischen Struktur genutzt. Eine Hierarchie wird mithilfe eines „/“-Symbols dargestellt. Um zu signalisieren, dass diese Struktur in der angegebenen URN

Ressource genutzt wird, beginnt diese mit „:/" nach der URN Adresse. Die Namensgebung für die obersten Strukturen der Domänen-Registry sind hierbei vorgegeben. Die Rollen, welche innerhalb dieser Domäne definiert werden, befinden sich in der Struktur „Rollen“ und werden innerhalb einer AutomationML Rollenklassenbibliothek verwaltet. Diese erhält hierdurch ihren eindeutigen Namen „Universität:ISW://Rollen“, der sich aus ihrer Domäne und dem Strukturnamen zusammensetzt.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <CAEXFile FileName="Domain-Registry-Example-Mit-Kontent.aml"
3  SchemaVersion="2.15"
4  xsi:noNamespaceSchemaLocation="CAEX_ClassModel_V2.15.xsd"
5  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
6    <AdditionalInformation>
7      <WriterHeader>
8        <WriterName>Matthias Strljic</WriterName>
9        <WriterID>916578CA-FE0D-474E-A4FC-9E1719892369</WriterID>
10       <WriterVendor>ISW</WriterVendor>
11       <WriterVendorURL>http://www.isw.uni-stuttgart.de/</WriterVendorURL>
12       <WriterVersion>4.3.11.0</WriterVersion>
13       <WriterRelease>4.3.11.0</WriterRelease>
14       <LastWritingDateTime>2016-09-05T14:39:40.131952+02:00</LastWritingDateTime>
15       <WriterProjectTitle>Masterarbeit</WriterProjectTitle>
16       <WriterProjectID>1337</WriterProjectID>
17     </WriterHeader>
18   </AdditionalInformation>
19   <AdditionalInformation AutomationMLVersion="2.0" />
20   <ExternalReference Path="Universität:ISG" Alias="ExternalRef" />
21   <InterfaceClassLib Name="Universität:ISW://Interfaces">
22     <Version>1.0.0</Version>
23   </InterfaceClassLib>
24   <RoleClassLib Name="Universität:ISW://Rollen">
25     <Version>1.0.0</Version>
26   </RoleClassLib>
27   <SystemUnitClassLib Name="Universität:ISW://Services">
28     <Version>1.0.0</Version>
29   </SystemUnitClassLib>
30   <SystemUnitClassLib Name="Universität:ISW://Komponenten">
31     <Version>1.0.0</Version>
32   </SystemUnitClassLib>
33   <SystemUnitClassLib Name="Universität:ISW://ServiceEndpunkte">
34     <Version>1.0.0</Version>
35   </SystemUnitClassLib>
36   <SystemUnitClassLib Name="Universität:ISW://Assets">
37     <Version>1.0.0</Version>
38   </SystemUnitClassLib>
39 </CAEXFile>

```

ABBILDUNG 48 DOMÄNEN-REGISTRY BESCHREIBUNG

Definierte Interfaces werden innerhalb der Struktur Interfaces aufbewahrt. Diese Struktur stellt eine AutomationML Interfaceklassenbibliothek dar und besitzt den eindeutigen Namen „Universität:ISW://Interfaces“. Dieser setzt sich aus der Domäne und dem Namen der eigenen Struktur zusammen. Die vier Systemklassenbibliotheken enthalten die für den jeweiligen Kontext definierten abstrakten Beschreibungen von Elementen. Hierbei enthält die Bibliothek mit dem Namen „Universität:ISW://Services“ die Service Definitionen, welche innerhalb dieser Domäne definiert werden. Die „Universität:ISW://Komponenten“ Bibliothek enthält alle abstrakten Beschreibungen von I4.0-Komponenten, welche in dieser Domäne definiert wurden.

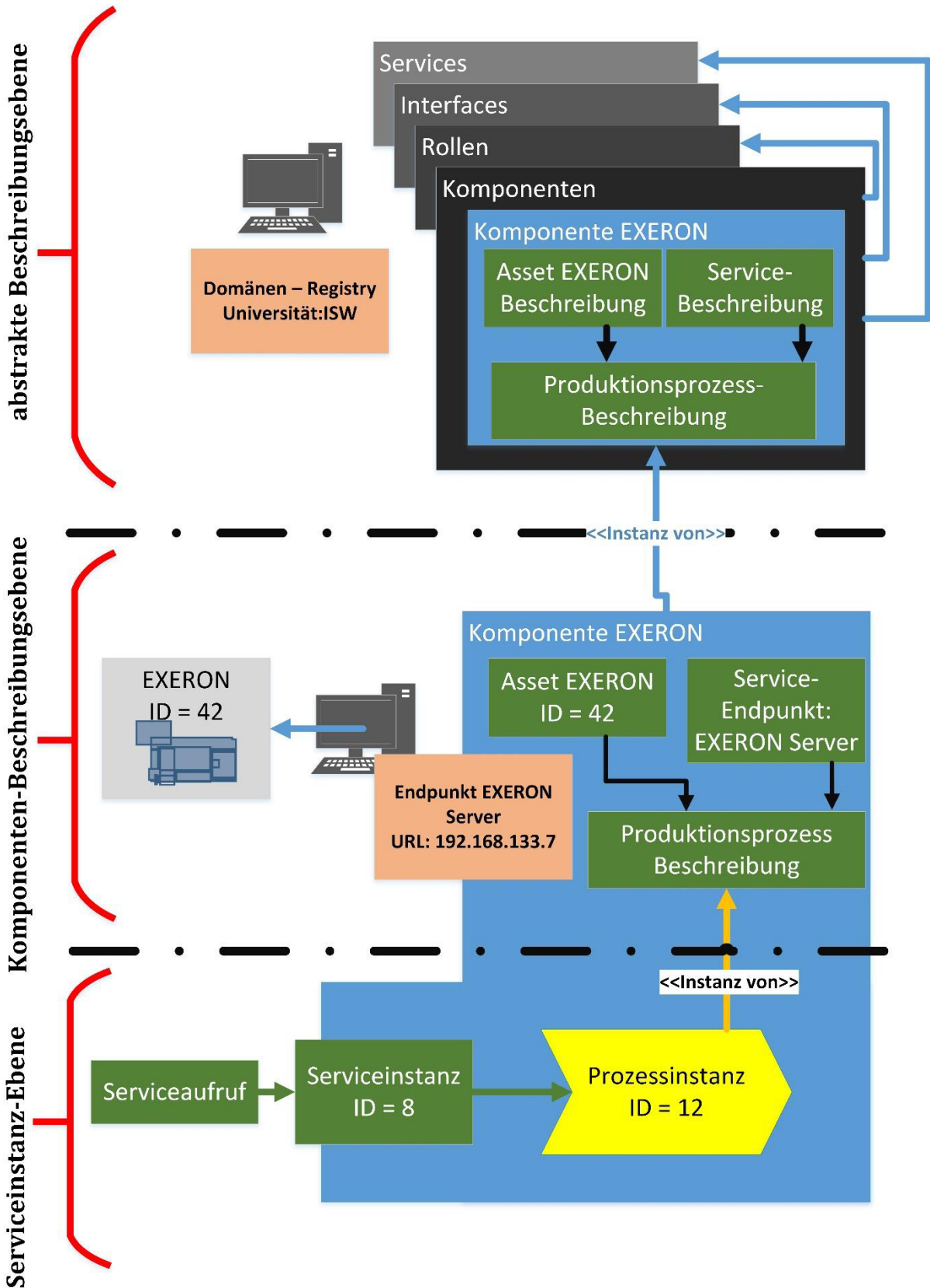


ABBILDUNG 49 BESCHREIBUNGSEBENEN

Die „Universität:ISW://Assets“ Bibliothek enthält ebenfalls Beschreibungen, in diesem Fall von Assets. Die „Universität:ISW://ServiceEndpunkte“ ist hierbei die einzige Bibliothek, welche keine reinen Beschreibungen enthält, sondern die Endpunkte von Services sowie von Komponenten. Diese repräsentieren I4.0-Komponenten und deren Services, welche auf einem System betrieben werden und über die im Endpunkt enthaltene URL erreichbar sind. Diese Elemente werden weiter im folgenden Kapitel erläutert. Zusätzlich existieren externe Referenzen, welche auf die Abhängigkeiten von Beschreibungen aus weiteren Domänen hinweist. Eine solche Referenz wurde hierfür eingefügt. Sie erhält den Alias „ExternalRef“ und referenziert auf die Beschreibungen aus der Domäne „Universität:ISG“. Dies bedeutet, dass für die Vollständigkeit die Bibliotheken von dieser Domänen-Registry zusätzlich heruntergeladen werden müssen. Diese Beschreibung ist wie die oben beschriebene Domänen-Registry aufgebaut, mit dem Unterschied, dass die Namen an die Domäne „Universität:ISG“ gebunden sind. Die Beschreibung einer Domänen-Registry ist eine von drei Betriebsebenen, welche im Betrieb einer I4.0-Komponente koexistieren. Diese werden in *Abbildung 49* abgebildet. Hierbei stellt die abstrakte Beschreibungsebene die Ebene für die Domänen-Registry Beschreibung dar. Hierin werden alle Assets-, Service- und I4.0-Komponentenbeschreibungen, welche für eine Domäne wichtig sind, zusammengeführt und definiert. Unterhalb dieser liegt die Komponenten-Beschreibungsebene, die eine Komponente im Betrieb darstellt. Diese Beschreibung stellt eine hybride Mischung aus Instanz- und Beschreibungselementen dar. Sie beinhaltet Prozess- und Service-Beschreibungen sowie Instanzen von Assets. Die wichtige Rahmenbedingung hierbei ist, dass diese Elemente der Beschreibung einer I4.0-Komponente aus ihrer Beschreibung der jeweiligen Domäne entspricht. Dies bedeutet, dass die in der abstrakten Beschreibung einer Komponente mit den Instanzen der jeweiligen Assets ersetzt werden. Diese Instanziierung muss der abstrakten Beschreibung folgen. Hierfür werden die abstrakten AutomationML System-Unit-Klassen instanziiert in Internal Elements, welche von diesen Klassen erben. Die letzte Ebene stellt die Serviceinstanz-Ebene dar. Diese umfasst eine reine Instanz-Umgebung und ist selbst noch Teil der I4.0-Komponenten-Laufzeitumgebung. In dieser Ebene befinden sich alle Instanziierungen eines Service. Alle für dessen Nutzung nötigen Asset-Instanzen werden hierbei als Abbild dieser in die Instanz mit einer neuen ID mit hineinkopiert. Diese Abbilder sind keine vollständigen Kopien, sondern beinhalten nur die genutzten Interfaces und internen Assets, welche von der Ausführung genutzt werden. Diese Abbilder erhalten ein „RefID“ Attribut, um auf ihre Ursprungsinstanz zu referenzieren. Ausgehend von dieser Domänen - Registry werden alle benötigten Ressourcen bereitgestellt, welche für die Nutzung und das Verständnis dieser Beschreibungen benötigt werden. Dies umfasst hiermit die Beschreibungen von I4.0-Komponenten, Assets und Services. Für die Nutzung stellt diese Domänen - Registry Dienste bereit, die das Durchsuchen der Registry, das Erstellen neuer Einträge sowie das Löschen von Einträgen umfasst. Wie diese Dienste aufgebaut sind, ist nicht Teil dieser Arbeit, sie werden jedoch für die Nutzung vorausgesetzt. Die Datenhaltung einer I4.0-Komponente kann im gesamten wiederum auch in einer AutomationML-Datei dargestellt werden. So besteht eine I4.0-Komponente aus einer AutomationML Instanz-Hierarchie und einer AutomationML System-Unit-Klassenbibliothek. Die System-Unit-Klassenbibliothek stellt die Komponenten-Beschreibungsebene dar und besitzt innerhalb der *Abbildung 50* den Namen „Universität:ISW://EXERON/Definition“. Der Name leitet sich von dem Namen der Komponente und der Domäne ab, in welcher diese betrieben wird. Diese Komponente wird innerhalb der Domäne „Universität:ISW“ betrieben und trägt den Namen „EXERON“. Die tiefere Struktur „Definition“ symbolisiert die Komponenten-Beschreibungsebene, welche das beschreibende Element „EXERON“ als System-Unit-Klasse enthält. Dies geschieht simultan mit der Instanz-Hierarchie, welche die tiefere Struktur „Instanz“ darstellt und somit den Namen „Universität:ISW://EXERON/Instanz“ trägt. Alle Instanz-Objekte, die zu der Serviceinstanz-Ebene

gehören, werden innerhalb dieser Hierarchie dargestellt. Jedes Element kann hierbei einzeln aus der I4.0-Komponente ausgelesen werden. Die Darstellung, innerhalb einer AutomationML-Datei, spiegelt die Rahmenbedingungen wieder. Jegliche Manipulation oder Erstellung von Objekten und Beschreibungen folgt der Definition als würde sie in einer solchen AutomationML-Datei geschehen.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <CAEXFile FileName="Empty-Komponent-EXERON.aml"
3  SchemaVersion="2.15"
4  xsi:noNamespaceSchemaLocation="CAEX_ClassModel_V2.15.xsd"
5  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
6    <AdditionalInformation>
7      <WriterHeader>
8        <WriterName>Matthias Strljic</WriterName>
9        <WriterID>916578CA-FE0D-474E-A4FC-9E1719892369</WriterID>
10       <WriterVendor>ISW</WriterVendor>
11       <WriterVendorURL>http://www.isw.uni-stuttgart.de/</WriterVendorURL>
12       <WriterVersion>4.3.11.0</WriterVersion>
13       <WriterRelease>4.3.11.0</WriterRelease>
14       <LastWritingDateTime>2016-09-06T09:41:27.8812374+02:00</LastWritingDateTime>
15       <WriterProjectTitle>Masterarbeit</WriterProjectTitle>
16       <WriterProjectID>1337</WriterProjectID>
17     </WriterHeader>
18   </AdditionalInformation>
19   <AdditionalInformation AutomationMLVersion="2.0" />
20   <InstanceHierarchy Name="Universität:ISW://EXERON/Instanz">
21     <Version>1.0.0</Version>
22   </InstanceHierarchy>
23   <SystemUnitClassLib Name="Universität:ISW://EXERON/Definition">
24     <Version>1.0.0</Version>
25     <SystemUnitClass Name="EXERON" />
26   </SystemUnitClassLib>
27 </CAEXFile>

```

ABBILDUNG 50 INITIALE KOMPONENTE

Für die Datenhaltung wird abschließend benötigt, wie die Plattformservices einer I4.0-Komponente auf deren Elemente Einfluss nehmen. Dies enthält Rahmenbedingungen für die Manipulation dieser Elemente. Hierfür werden in der folgenden Auflistung die jeweiligen Dienste aufgezählt und ihre definierte Aufgabe erläutert.

Plattformservice – Lesen stellt einen reinen Zugriff auf die Elemente einer I4.0-Komponente dar. Es können hiermit alle Elemente innerhalb der I4.0-Komponente eingesehen werden. Dies gilt solange der Zugreifende die benötigten Berechtigungen besitzt. Es können Elemente aus der Komponenten-Beschreibungsebene sowie aus der Serviceinstanz-Ebene ausgelesen werden.

Plattformservice – Schreiben manipuliert Attribute von Instanz-Assets. Es ist diesem Dienst nicht gestattet Attribute von Instanz-Objekten aus der Serviceinstanz-Ebene zu überschreiben, da diese nur von der I4.0-Komponente selbst erstellt und verändert werden. Eine Manipulation von außen würde eine Werteänderung darstellen, welche nicht durch den Prozess beschrieben ist und somit nicht eindeutig definiert ist. Zusätzlich müssen alle geschriebenen Attribute der Beschreibung eines Assets folgen. Das bedeutet, dass z.B. ein Attribut vom Typ „URL“ auch nur durch eine URL beschrieben werden kann. Der **Plattformservice – Schreiben** beschränkt sich somit auf die Manipulation der Komponenten-Beschreibungsebene.

Plattformservice – Erstellen generiert neue Elemente innerhalb der Komponenten-Beschreibungsebene. Dies bezieht sich auf die Instanziierung von Asset-Beschreibungen oder neuer Asset-Instanzen innerhalb eines Assets in der Semantik seiner Beschreibung. Dies könnte z.B. das Hinzufügen einer neuen OPC UA Variablen innerhalb eines OPC UA Server Assets sein. Die Instanziierung von Asset-Beschreibungen, welche nicht innerhalb eines Assets stattfindet ist nicht gestattet. Diese Instanziierung erfolgt über den zusätzlichen **Plattformservice - Instanzieren**. Der Grund warum dieser Dienst keinen Zugriff auf die Serviceinstanz-Ebene besitzt ist derselbe wie für den **Plattformservice – Schreiben**.

Plattformservice – Löschen entfernt Elemente aus der Komponenten-Beschreibungsebene. Es ist nur gestattet Asset-Beschreibungen und Elemente innerhalb von Asset-Instanzen zu entfernen. Eine solche Löschung muss hierbei der Semantik der Asset-Instanz entsprechen. Es ist nicht zulässig eine vollständige Asset-Beschreibung zu entfernen, welche die Wurzelstruktur des Assets darstellt. Dies ist nur durch den **Plattformservice – Instanzieren** möglich. Dieser Dienst besitzt ebenfalls keinen Zugriff auf die Serviceinstanz-Ebene, da es sich um einen manipulierenden Dienst handelt.

Plattformservice – Abonnieren stellt einen Dienst dar, welcher genutzt wird um externe Assets in eine I4.0-Komponente zu spiegeln. Das Abonnieren kann nur eine existierende Asset-Instanz ersetzen. Dies erfolgt nach dem Schema als würde die Komponente selbst den Wert auslesen und diesen dann auf ein internes Element schreiben wollen. Der Vorteil dieses Dienstes ist es, dass dieser auf das Erstellen, Ändern oder eine zyklische Bedingung konfiguriert werden kann. Es muss dabei die Definition von externen Assets mit AutomationML aus *Kapitel 4.3* eingehalten werden

Plattformservice – Browse stellt die Möglichkeit dar die gesamte I4.0-Komponente auszulesen. Dieser Service soll es ermöglichen sich durch die Hierarchie der Komponenten-Beschreibungsebene sowie der Serviceinstanz-Ebene zu navigieren. Dieser Dienst ist nötig, da nicht von vornherein klar ist wie die Serviceinstanz-Ebene aufgebaut ist und die darin enthaltenen Elemente benannt wurden. Ein zusätzlicher Vorteil ist eine strukturierte Navigation, ohne den Bedarf eine Struktur komplett auszulesen und somit Bandbreite bei der Übertragung einzubüßen für Daten, welche nicht von Interesse sind. Dieser Dienst ist in der Lage die Komponenten-Beschreibungsebene sowie die Serviceinstanz-Ebene zu durchsuchen.

Plattformservice – Instanzieren ist ein zusätzlicher Dienst, welcher benötigt wird um eine Asset-Beschreibung in eine Instanz umzuformen. Dies könnte ebenso durch die Löschung und Erstellung durch die jeweiligen Dienste erfolgen, würde hierbei aber nicht wohldefiniert sein. Dies resultiert aus der Tatsache, dass zwar ein externer Dienst dazu verpflichtet sein kann diese Aktionsfolge durchzuführen, dieser aber nach der Löschung unwiderruflich vernichtet werden kann und somit keine Erstellung folgt. Würde eine solche Interaktion zugelassen, könne eine I4.0-Komponente in einen nicht definierten Zustand versetzt werden. Dieser Dienst kapselt diese Interaktion in einem Aufruf. Er löscht zunächst die Asset-Beschreibung und instanziiert die Asset-Instanz nach dessen korrekter Semantik aus der abstrakten Beschreibungsebene. Dieser Dienst kann nur Instanzen innerhalb der Komponenten-Beschreibungsebene erstellen. Der Grund hierfür ist zum einen, dass es sich um einen manipulierenden Dienst handelt und zum anderen, dass innerhalb dieser Ebene keine Beschreibung existiert, welche instanziiert werden kann.

4.3 REALISIERUNG MIT AUTOMATIONML

Dieses Kapitel umfasst das Mapping aller vom Datenmodell, aus *Kapitel 3.3*, definierten Entitäten in das AutomationML Format. Alle Definitionen erfolgen innerhalb einer definierten AutomationML System Klassen, Rollenklassen und Interfaceklassenbibliothek, welche ein Standard Import ist, um modellierte I4.0-Komponenten und deren Assets zu interpretieren. Ebenso werden die von der *AutomationML Initiative* standardisierten Rollen und Interface Bibliotheken vorausgesetzt. Diese werden für die optimale Entwicklung von Assets benötigt. Für eine verbesserte Übersicht folgt die Struktur dieses Kapitels der Struktur des *Kapitels 3.3*. Die aktuelle Version dieses Importes beschränkt sich auf die Nutzung der Rollen und Interfaceklassenbibliothek. Die folgenden Kapitel beschreiben jeweils die Anpassungen, welche vorgenommen werden mussten, um das Datenmodell in AutomationML umzusetzen und führen für ein verbessertes Verständnis Beispiele oder das direkte Mapping auf. Jede Rolle dient als abstrakte Definition eines Unterelements. Wird eine Rolle hierarchisch unter einer Rolle definiert, ist es ebenfalls nur erlaubt diese Rolle einem Element zuzuweisen, welches unterhalb eines Elementes liegt, das die überliegende Rolle erfüllt. Eine Ausnahme von diesen Vorgaben werden als Abweichungen hervorgehoben.

4.3.1 DOMÄNE

Eine Domäne selbst stellt über die Datenhaltung mit AutomationML ein rein logisches Objekt dar. Es wird über seine URN repräsentiert und ist über die zuständige Domänen-Registry erreichbar. Die innerhalb einer Domäne liegenden Objekte werden in den innerhalb von *Kapitel 4.2* vorgestellten Registry AutomationML Bibliotheken dargestellt. Die Servicebeschreibung befindet sich innerhalb der System-Unit-Klassenbibliothek „Services“ einer Domänen-Registry. Eine Beschreibung eines Endpunktes findet innerhalb der Bibliothek „ServiceEndpunkte“ statt. Die *Abbildung 51* beschreibt wie der Service „BeispielService“ innerhalb einer Domäne dargestellt werden kann. Zusätzlich wird dieser Service auf einem Endpunkt mit dem Namen „Maschine1“ betrieben. Die hierbei verwendeten Rollen werden in *Kapitel 4.3.3* und *Kapitel 4.3.2* näher besprochen und dienen der Identifikation der Bedeutung eines Elements. Der „BeispielService“ ist minimalistisch aufgebaut und enthält keine Antwortnachricht. Er enthält eine Servicefunktion mit dem Namen „BeispielFunktion“, welche über das Protokoll „HTTP“ angesprochen werden kann. Er wird über die FAN „AufrufNachricht“ definiert, welche sich innerhalb des Nachrichtenteiles „Body“ befindet und das „BeispielFeld“ als Feld beinhaltet. Diese Definition befindet sich in der System-Unit-Klassenbibliothek „Services“. Das ein Service auf einem Endpunkt betrieben wird, bildet das Element „LaufenderService“ ab. Es erbt von dem oben definierten „BeispielService“. Dies definiert, dass es sich um solch einen laufenden Service handelt. Die QoS, welche ein Service bereitstellt, werden über die Zuweisung als Rolle auf diese Entität definiert. So besitzt der „LaufendeService“ eine QoS-Erreichbarkeit von 99%. Der Endpunkt auf welchem dieser Service erreichbar ist, stellen Elemente mit der Rolle „Endpunkt“ dar. In diesem Beispiel handelt es sich um einen Endpunkt mit dem Namen „Maschine_1“. Er stammt aus der Domäne „Universität:ISW“ und ist dort selbst das Element „Maschine1“. Dies führt zu der gesamten URN „Universität:ISW://Maschine1“. Der Endpunkt selbst ist unter der URL „http://isw.universität-stuttgart.de/maschine1“ erreichbar. Ein solches Element zur Beschreibung von laufenden Services besitzt diesen definierten Aufbau. Die Beschreibung von erreichbaren Komponenten erfolgt simultan, mit der Ausnahme, dass diese keine QoS-Rollen erhalten können.

```

29 <SystemUnitClassLib Name="Universität:ISW://Services">
30   <Version>1.0.0</Version>
31   <SystemUnitClass Name="BeispielService">
32     <SystemUnitClass Name="BeispielFunktion">
33       <Attribute Name="Protokoll"
34         AttributeDataType="xs:normalizedString">
35         <Value>HTTP</Value>
36       </Attribute>
37       <SupportedRoleClass
38         RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]/Service/Service Funktion" />
39       <SystemUnitClass Name="AufrufNachricht">
40         <SupportedRoleClass
41           RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]
42             /Service/Service Funktion/Funktionsaufrufnachricht" />
43         <SystemUnitClass Name="Body">
44           <SupportedRoleClass
45             RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]
46               /Service/Service Funktion/Nachricht/Nachrichtenteil" />
47           <SystemUnitClass Name="BeispielFeld">
48             <SupportedRoleClass
49               RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]
50                 /Service/Service Funktion/Nachricht/Nachrichtenteil/Nachrichtenfeld" />
51             </SystemUnitClass>
52           </SystemUnitClass>
53         </SystemUnitClass>
54       </SystemUnitClass>
55     </SystemUnitClass>
56 </SystemUnitClassLib>
57 <SystemUnitClassLib Name="Universität:ISW://ServiceEndpunkte">
58   <Version>1.0.0</Version>
59   <SystemUnitClass Name="LaufenderService"
60     RefBaseClassPath="[Universität:ISW://Services]/BeispielService">
61     <Attribute Name="Erreichbarkeit_QoS" AttributeDataType="xs:float">
62     <Value>0.99</Value>
63   </Attribute>
64   <SupportedRoleClass
65     RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]
66       /Service/Quality of Service/Erreichbarkeit" />
67   <SystemUnitClass Name="Maschine_1">
68     <Attribute Name="Adresse" AttributeDataType="xs:anyURI">
69     <Value>http://isw.universität-stuttgart.de/maschine1</Value>
70   </Attribute>
71     <Attribute Name="Domäne" AttributeDataType="xs:anyURI">
72     <Value>Universität:ISW://Maschine1</Value>
73   </Attribute>
74     <SupportedRoleClass RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]/Enpunkt" />
75   </SystemUnitClass>
76 </SystemUnitClass>
77 </SystemUnitClassLib>

```

ABBILDUNG 51 BEISPIEL SERVICEBESCHREIBUNG MIT ENDPUNKT

4.3.2 SERVICEBESCHREIBUNG

Die Servicebeschreibung besteht aus Rollen für die jeweilig zu repräsentierende Entität des Datenmodells mit seinen zugehörigen Attributen. Zusätzlich wird noch das „ServiceFunktionTriggerInterface“ erstellt. Da in AutomationML jegliche Form von Entitäten keine direkten Relationen besitzen kann, werden diese über die Verbindung von Interfaces mithilfe „InternalLink“- oder „AbstractInternalLink“-Elemente dargestellt. Das zusätzlich erstellte Interface wird benötigt, um dieses an Nachrichten des Service zu binden, um diese wiederum mit Prozessen zu verbinden. Diese Verbindung signalisiert das Binden eines Prozesses an den jeweiligen Service und dessen Servicefunktion. Eine solche Bindung hat den Effekt, dass wenn eine Servicefunktion aufgerufen und somit instanziiert wird, auch der damit

verbundene Prozess instanziiert und ausgeführt wird. Eine beispielhafte Erstellung eines Service befindet sich in *Kapitel 4.3.1* und wurde bereits besprochen. *Abbildung 52* illustriert den Ausschnitt für die Definition der Rollen. Am unteren Ende wurde zusätzlich die Interface-Definition angehängt, welche aus der Interfaceklassenbibliothek stammt. Das gesamte Mapping ist der Abbildung zu entnehmen. Anpassungen beziehen sich in diesem Bereich auf die Funktionsresultatnachricht. Diese wurde nach den AutomationML Richtlinien [31] für ihren „ResultatTyp“ mit „Enum“-Restriktionen versehen, welche die Art von Rückgabe für eine FRN angibt. Wenn es sich hierbei um ein „RequestResultat“ handelt, werden die Attribute für dieses innerhalb des „RequestResultat“ Attributes verschachtelt. Dieses hält die beiden Unterattribute **ID** und **Adresse**, um das Resultat zu referenzieren. QoS wurde für die jeweilige QoS als separate Rolle definiert. Die semantische Bedeutung wurde jedoch über die Vererbung von „Quality of Service“ gekennzeichnet. Die Zuweisung erfolgt dadurch, dass die jeweilige QoS-Rolle dem Service direkt zugewiesen wird und dieser dann die nötigen Attribute wie z.B. Erreichbarkeit erhält.

```

2   <RoleClass Name="Service">
3     <RoleClass Name="Service Funktion">
4       <Attribute Name="Protokoll"
5         AttributeDataType="xs:normalizedString" />
6       <RoleClass Name="Nachricht">
7         <RoleClass Name="Nachrichtenteil">
8           <RoleClass Name="Nachrichtenfeld">
9             <Attribute Name="Typ"
10              AttributeDataType="xs:normalizedString" />
11            <Attribute Name="Wert"
12              AttributeDataType="xs:normalizedString" />
13          </RoleClass>
14        </RoleClass>
15      </RoleClass>
16      <RoleClass Name="Funktionsaufrufnachricht"
17        RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]
18        /Service/Service Funktion/Nachricht">
19        <Attribute Name="ReplyTo"
20          AttributeDataType="xs:anyURI" />
21      </RoleClass>
22      <RoleClass Name="Funktionsresultatnachricht"
23        RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]
24        /Service/Service Funktion/Nachricht">
25        <Attribute Name="ResultatTyp">
26          <Constraint Name="ResultatTypen">
27            <NominalScaledType>
28              <RequiredValue>RESULTAT</RequiredValue>
29              <RequiredValue>REQUEST_RESULTAT</RequiredValue>
30              <RequiredValue>EVENT</RequiredValue>
31            </NominalScaledType>
32          </Constraint>
33        </Attribute>
34        <Attribute Name="RequestResultat">
35          <Attribute Name="ID" AttributeDataType="xs:normalizedString" />
36          <Attribute Name="Adresse" AttributeDataType="xs:anyURI" />
37        </Attribute>
38      </RoleClass>

```

```

39     <RoleClass Name="Funktionsfehlernachricht"
40       RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Service/Service Funktion/Nachricht">
41       </RoleClass>
42 </RoleClass>
43 <RoleClass Name="Quality of Service">
44   <RoleClass Name="Erreichbarkeit"
45     RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Service/Quality of Service">
46     <Attribute Name="Erreichbarkeit_QoS"
47       AttributeDataType="xs:float"></Attribute>
48   </RoleClass>
49   <RoleClass Name="Integrität"
50     RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Service/Quality of Service">
51     <Attribute Name="Methode_Integrität"
52       AttributeDataType="xs:normalizedString" />
53   </RoleClass>
54   <RoleClass Name="Performanz"
55     RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Service/Quality of Service">
56     <Attribute Name="Latenz" AttributeDataType="xs:normalizedString" />
57     <Attribute Name="Durchsatz" AttributeDataType="xs:float" />
58   </RoleClass>
59   <RoleClass Name="Zuverlässigkeit"
60     RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Service/Quality of Service">
61     <Attribute Name="MTTR" AttributeDataType="xs:unsignedLong" />
62   </RoleClass>
63   <RoleClass Name="Sicherheit"
64     RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Service/Quality of Service">
65     <Attribute Name="Mechanik" AttributeDataType="xs:normalizedString" />
66   </RoleClass>
67 </RoleClass>
68 </RoleClass>
69 <InterfaceClassLib Name="I4.0_Konzept:Base://Interfacelib">
70   <Version>1.0.0</Version>
71   <InterfaceClass Name="ServiceFunktionTriggerInterface" />
72   <InterfaceClass....>

```

ABBILDUNG 52 SERVICEBESCHREIBUNG AUTOMATIONML

4.3.3 PROZESSBESCHREIBUNG

Die gesamte Prozessbeschreibung besteht aus einem fast direkten Mapping der vorgestellten Entitäten aus Kapitel 3.3.3. Für die Generalisierung von Aktivitäten, Gates und Events wurden äquivalent benannte Vaterrollen erstellt, von welchen die jeweiligen Rollen erben. Dies ermöglicht es weitere Rollen für diesen Bereich als Standard zu definieren, ohne die Semantik des Modells zu verändern. Zusätzlich gilt dies auch für potentielle proprietäre Entwicklungen von Herstellern, welche eigene Rollen für ihre Domäne definieren können. Die Flow-Entität wurde durch die existierenden „InternalLink“-Elemente von AutomationML dargestellt. Da AutomationML keine „InternalLink“-Elemente zwischen abstrakten System-Unit-Klassen unterstützt, werden hierfür „AbstractInternalLink“-Elemente definiert. Diese agieren analog zu den AutomationML Elementen und dienen nur der Darstellung von Verbindungen, worin mindestens eine System-Unit-Klasse involviert ist. Für eine eindeutige Verbindung, mithilfe dieser ein Flow dargestellt wird, existieren die beiden AutomationML Interfaceklassen „MessageFlowInterface“ und „SequentialFlowInterface“. Diesen ist es gestattet an AutomationML Elemente angebracht zu werden, welchen eine Rolle aus dieser Prozessrollenbibliothek zugewiesen ist.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <RoleClass Name="Prozess">
3      <RoleClass Name="Gate">
4          <RoleClass Name="pGate"
5              RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Prozess/Gate" />
6          <RoleClass Name="oGate"
7              RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Prozess/Gate">
8              <Attribute Name="Bedingung" AttributeDataType="xs:string" />
9          </RoleClass>
10         <RoleClass Name="kGate"
11             RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Prozess/Gate">
12             <Attribute Name="Bedingung" AttributeDataType="xs:string" />
13         </RoleClass>
14     </RoleClass>
15     <RoleClass Name="Aktivität">
16         <RoleClass Name="FieldAssigner"
17             RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Prozess/Aktivität">
18             <Attribute Name="FromFields" />
19             <Attribute Name="ToFields" />
20         </RoleClass>
21     </RoleClass>
22     <RoleClass Name="Event">
23         <RoleClass Name="End_Event"
24             RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Prozess/Event">
25             <RoleClass Name="Message_Request"
26                 RefBaseClassPath=
27                 "[I4.0_Konzept:Base://RoleLib]/Prozess/Event/End_Event" />
28         </RoleClass>
29         <RoleClass Name="Start_Event"
30             RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Prozess/Event">
31             <RoleClass Name="Response_Message"
32                 RefBaseClassPath=
33                 "[I4.0_Konzept:Base://RoleLib]/Prozess/Event/Start_Event" />
34         </RoleClass>
35         <RoleClass Name="Zwischen_Event"
36             RefBaseClassPath="[I4.0_Konzept:Base://RoleLib]/Prozess/Event" />
37     </RoleClass>
38     <RoleClass Name="Daten_Quelle" />
39 </RoleClass>
40 <InterfaceClass Name="ProzessInterfaces">
41     <InterfaceClass Name="MessageFlowInterface"
42         RefBaseClassPath="[I4.0_Konzept:Base://InterfaceLib]/ProzessInterfaces" />
43     <InterfaceClass Name="SequentialFlowInterface"
44         RefBaseClassPath="[I4.0_Konzept:Base://InterfaceLib]/ProzessInterfaces" />
45     <InterfaceClass Name="ProcessEndeInterface"
46         RefBaseClassPath="[I4.0_Konzept:Base://InterfaceLib]/ProzessInterfaces" />
47 </InterfaceClass>

```

ABBILDUNG 53 PROZESSBESCHREIBUNG AUTOMATIONML

Die Verbindung zwischen diesen Interfaces darf nur homogen geschehen, damit ist die Verbindung eines „MessageFlowInterface“ mit einem „SequentialFlowInterface“ nicht erlaubt. Bei der Verbindung von „MessageFlowInterface“-Elementen muss zusätzlich das Attribut „MessageRef“ bestehen, um auf die zugehörige Nachricht zu referenzieren. Die Ausrichtung von einem Flow richtet sich nach der

Verbindungsrichtung des „InternalLink“-Elements oder des „AbstractInternalLink“-Elements. Für die korrekte Aneinanderreihung von Prozessen ist zusätzlich ein „ProzessEndeInterface“ hinzugefügt worden. Dieses kann genutzt werden, um Prozesse miteinander zu verbinden und somit die Ausführung eines Folgeprozesses zu symbolisieren. Prozesse und Aktivitäten ist es zudem gestattet jegliche nicht Prozessinterfaces zu nutzen. Die Nutzung eines solchen als Asset-Interface bezeichneten Interfaces benötigt hierbei eine eindeutige Spezifizierung. Eine solche Spezifizierung ist nicht Teil dieser Arbeit, erfolgt aber simultan zu einer normalen Interfacedefinition in AutomationML. Das vollständige Mapping aller Prozessentitäten wird in *Abbildung 53* abgebildet. Die nötigen AutomationML Rollenklassen sowie Interfaceklassen wurden, für eine verbesserte Übersicht, aus den jeweiligen Bibliotheken in eine Datei kopiert.

4.3.4 ASSET

Da die Beschreibung von Assets bereits an AutomationML angelehnt wurde, werden Teile der Asset definierenden Entitäten direkt mit AutomationML Entitäten abgebildet. Die Beschränkung hierbei ist lediglich, dass jegliche Asset-Modellierung innerhalb eines Elementes stattfinden muss, das die Rolle „Asset“ erfüllt. So werden „Link“-Entitäten durch AutomationML „InternalLink“-Elemente dargestellt. Interface-Entitäten werden über AutomationML Interfaces dargestellt, welche über die „InternalLink“-Elemente verbunden werden können. Rollen und Attribute werden über ihre gleichnamigen AutomationML Gegenstücke abgebildet. Eine externe Entität oder ein externes Asset werden über die Rolle „ExterneReferenz“ abgebildet, dessen Definition in *Abbildung 54* dargestellt wird. Diese beinhaltet das Attribut „Ressource“, welches durch die eindeutigen URN auf die Ursprungsressource referenziert.

```
2 <RoleClass Name="ExterneReferenz">
3   <Attribute Name="Ressource" AttributeDataType="xs:anyURI" />
4 </RoleClass>
```

ABBILDUNG 54 EXTERNE REFERENZ

Diese Rolle wird direkt auf das jeweilige Asset gelegt, um dessen externe Deutung zu repräsentieren. Die „ExterneReferenz“-Rolle ist mit den „Quality of Presentation“-Rollen die einzigen Rollen, welche innerhalb der Komponenten-Beschreibungsebene abweichend von dessen abstrakten Beschreibung erlaubt ist. Diese Abweichung ist nötig, da bei der Beschreibung einer Komponente nicht klar sein muss, ob ein jeweiliges Asset zu einem Zeitpunkt an einem bestimmten Ort sich befindet. Betrachtet man z.B. eine G-Code Datei, welche von einer NC-Maschine eingelesen werden soll, wird klar, dass es nicht relevant ist woher diese Datei stammt. Die Tatsache genügt, dass diese definiert abgerufen werden kann. Es ist innerhalb der Beschreibung nur relevant, um was für einen Typ von Asset es sich handelt. Die Herkunft eines solchen Assets wird erst mit der Nutzung seiner Instanz relevant und diese kann sich zudem innerhalb einer anderen I4.0-Komponente befinden. Darum muss es möglich sein darzustellen, dass eine I4.0-Komponente dieses Asset von einer anderen I4.0-Komponente bezieht. Die „Quality of Presentation“(QoP) wird hierbei wie die QoS aus *Kapitel 4.3.2* durch eine Vaterrollenklasse repräsentiert und ist in *Abbildung 55* dargestellt. Die drei QoP Eigenschaften „Validität“, „Latenz“ und „Detaillevel“ werden als separate Rolle realisiert, welche von dieser Vaterrolle erben. Sie können jedem Element innerhalb eines Assets zugewiesen werden sowie dem gesamten Asset selbst. Es ist hierbei wichtig zu beachten, dass verschachtelte QoP sich aufeinander auswirken. Dies bezieht sich auf die beiden Eigenschaften „Validität“ und „Latenz“. Für die „Latenz“ gilt, dass diese generell für alle inneren Elemente gilt.

```

5   <RoleClass Name="Quality of Presentation">
6     <RoleClass Name="Validität">
7       <Attribute Name="TTL" AttributeDataType="xs:dateTime" />
8     </RoleClass>
9     <RoleClass Name="Latenz">
10      <Attribute Name="Latenz" AttributeDataType="xs:duration" />
11    </RoleClass>
12    <RoleClass Name="Detail_Level">
13      <Attribute Name="Detail_Level"
14        AttributeDataType="xs:normalizedString">
15        <Constraint Name="DL_Levels">
16          <NominalScaledType>
17            <RequiredValue>VOLLSTÄNDIG</RequiredValue>
18            <RequiredValue>PARTIAL</RequiredValue>
19            <RequiredValue>VERSTECKT</RequiredValue>
20          </NominalScaledType>
21        </Constraint>
22      </Attribute>
23    </RoleClass>
24  </RoleClass>

```

ABBILDUNG 55 QUALITY OF PRESENTATION

Wird eine weitere QoP-„Latenz“ einem innerem Element zugewiesen, addiert sich diese mit allen umschließenden QoP-„Latenz“ Elementen. Für die „Validität“ gilt, dass kein Element innerhalb eines Elementes mit dieser Eigenschaft eine kleinere Zeitspanne angeben kann als das umschließende QoP-„Validität“ Element. Denn sonst wäre durch ein innen liegendes invalides Element automatisch das umschließende Element zugleich invalide. Die QoP Eigenschaft „Detaillevel“ wurde mit der Restriktion „DL_Levels“ versehen, um das Attribut „Detail_Level“ auf diese Wertebelegungen zu beschränken. Die restlichen Entitäten werden in *Abbildung 56* dargestellt. Es handelt sich um die Elemente „Asset_Daten_Quelle“, „Controller“ und „DeepLink“. Die „Asset_Daten_Quelle“ ist die Rolle für ein Objekt in einem Asset, welches mit mindestens dem Attribut „Ressourcen_Adresse“ über eine URI auf eine Ressource referenziert, welche das Asset beschreibt. Es kann sich dabei z.B. um die Entwurfsskizze einer CAD-Datei handeln. Ein Element mit „DeepLink“ als Rolle verweist über seine Attributliste „Referenz_Links“ auf eine Gruppe an zugehörigen „InternalLink“-Elementen. Dies wird genutzt, um von einem anderem „InternalLink“-Element auf dieses „DeepLink“-Element zu referenzieren und somit komplexe zusammenhängende Relationen zu beschreiben. Dies könnte z.B. die logische Verbindung zu einer Maschine sein, welche weitere physische Verbindungen nutzt.

```

26  <RoleClass Name="Asset_Daten_Quelle">
27    <Attribute Name="Ressourcen_Adresse" AttributeDataType="xs:anyURI" />
28  </RoleClass>
29  <RoleClass Name="Controller" /></RoleClass>
30  <RoleClass Name="DeepLink">
31    <Attribute Name="Referenz_Links" />
32  </RoleClass>

```

ABBILDUNG 56 ASSET BESCHREIBUNGSELEMENTE

Das letzte Element stellt den Controller eines Assets bereit. Dieser ist dafür zuständig Interfaces und Services eines Assets nach außen zu bieten. Dies bedeutet, dass Services und Interfaces, welche über dieses Element dargestellt werden, für die Verwendung von Prozessen innerhalb der I4.0-Komponente zur Verfügung stehen. Eine solche Funktion könnte z.B. ein OPC UA Interface sein, welches von einem Server nach außen angeboten wird. Ein Asset kann hierbei mehrere Controller besitzen, um mehrfach angebotene Interfaces semantisch zu unterscheiden.

4.3.5 I4.0 KOMPONENTE

Die I4.0-Komponente besitzt selbst das Attribut Domäne, welches seine zugehörige Domäne widerspiegelt. Eine I4.0 Komponente besteht aus den vier Elementen Endpunkt, Service, Prozess und Asset. Es ist nicht gestattet weitere Elementtypen zu verwenden. Der Endpunkt stellt simultan zu Services eine logische Adresse dar, unter welcher eine I4.0-Komponente erreicht werden kann. Unter dieser Adresse können alle Plattformservices von einer I4.0-Komponente genutzt werden. Diese Endpunkte müssen nicht mit den Endpunkten der innerhalb einer I4.0-Komponente verwalteten Services übereinstimmen. Da AutomationML keine „InternalLink“-Elemente zwischen System-Unit-Klassen unterstützt, musste eine neue System-Unit-Klasse mit dem Namen „AbstractInternalLink“ erstellt werden. Diese erfüllt den Zweck abstrakte Verbindungen zwischen mindestens einem Interface einer System-Unit-Klasse zu repräsentieren. Wird die System-Unit-Klasse durch eine Instanz ersetzt, wird dieser „AbstractInternalLink“ zu einem normalen „InternalLink“. Der Aufbau von Endpunkte-Rollen und I4.0-Komponenten-Rollen befindet sich in *Abbildung 57*.

```
2 <RoleClass Name="I4.0 Komponente">
3   <Attribute Name="Domäne" AttributeDataType="xs:anyURI" />
4 </RoleClass>
5 <RoleClass Name="Enpunkt">
6   <Attribute Name="Adresse" AttributeDataType="xs:anyURI" />
7   <Attribute Name="Domäne" AttributeDataType="xs:anyURI" />
8 </RoleClass>
```

ABBILDUNG 57 I4.0-KOMPONENTE AUTOMATIONML

4.4 VALIDIERUNG ANHAND EINER BEISPIELHAFTEN UMSETZUNG

In diesem Unterkapitel wird eine beispielhafte Nutzung des Modells vorgeführt. Es wird ein Ausschnitt an Assets bezogen, welche aus einer Bachelorarbeit von Andreas Schütz entnommen wurden. Diese Nutzung soll somit als Validierung des Modells gelten und dessen Verwendung illustrieren. Da eine komplette Validierung mithilfe eines AutomationML Modells im Detail zu umfangreich in der Darstellung ist, wird über eine abstrakte Darstellung mit Auszügen aus der AutomationML Umsetzung illustriert. Die Umsetzung nutzt hierbei die Industrie 4.0 Terminologie sowie die Entitäten aus dem Datenmodell. Es handelt sich um die Erstellung eines Fahrradpedals als Produkt. Die für dieses Beispiel wichtigen Elemente stellen die Assets für die Herstellung des Korpus dieses Fahrradpedals dar. Es wurde die in *Abbildung 58* dargestellte CAD-Zeichnung dieses Pedals innerhalb des Produktengineering erstellt. Im Anschluss wurden Ablaufpläne für die Erstellung der Einzelteile angefertigt. Diese hatten als Resultat, dass dieses Pedal aus zwei separat gefertigten Teilen besteht. Es handelt sich hierbei um den Pedal-Korpus und eine Achse, welche durch eine Bohrung innerhalb des Korpus versenkt wird. Ausschlaggebend für dieses Beispiel sind die Schritte zur Erstellung des Korpus, die Darstellung der Assets innerhalb einer I4.0-Komponente und wie eine solche I4.0-Komponenten erstellt wird. Die Erstellung des Korpus wurde über einen CAM-Prozess mit der Software SolidWorks erstellt. Dies hat

zur Folge, dass der Korpus mittels 3 NC-Programmen gefertigt wird, welche sequentiell in einer CNC-Maschine vom Typ „Exeron HSC 600“ abgearbeitet wird. *Abbildung 59* zeigt im Unterpunkt *a)* den hierbei entstandenen Ablaufplan für die Produktion.

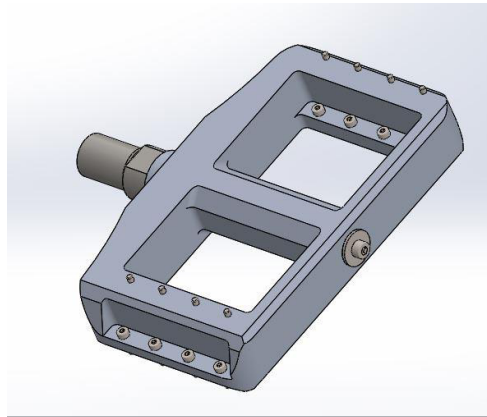


ABBILDUNG 58 FAHRRADPEDAL CAD

Für diesen CAM – Prozess wurde von einem Rohmaterialblock ausgegangen, welcher in *Abbildung 59* Unterpunkt *b)* dargestellt wird. Ausgehend von diesem Block wird zuerst die Kontur gefräst, im Anschluss die Bohrung für die Achse vollzogen und abschließend werden 16 Löcher mit ihren zugehörigen Gewinden erstellt. Für jedes dieser Schritte wird ein eigenes NC-Programm in G-Code erstellt. Die einzelnen NC-Programme werden visuell über die Unterpunkte *b)* bis *e)* innerhalb von *Abbildung 59* illustriert. Nach diesem CAM-Prozess werden die drei NC-Programme zusammen mit einem Rohmaterialblock zu der „Exeron HSC 600“ transportiert. Dort wird das Rohmaterial eingespannt und die einzelnen Programme sequentiell eingelesen und gestartet.

Es wurde im Folgenden auf das Fräsen des Korpus fokussiert. So wurde im Ablaufschritt „Kontur fräsen“ der resultierende Maschinen-Log ausgelesen und ein Schnappschuss von Echtzeitdaten aufgenommen. Der resultierende Maschinen-Log wird hierbei in *Abbildung 60* dargestellt und der Schnappschuss in *Abbildung 61*. Nach diesem Schritt ist die Kontur fertig und es könnte mit den weiteren zwei NC-Programmen fortgeschritten werden, welche einen ähnlichen Ablauf besitzen. Dieser beispielhafte Ausschnitt soll über die Verwendung von I4.0-Komponenten geschehen. Es wird hierbei versucht diese Arbeitsschritte abzubilden und in I4.0-Komponenten zu integrieren. Der Ablauf wird nicht weiter verändert oder mit zusätzlicher Funktionalität erweitert.

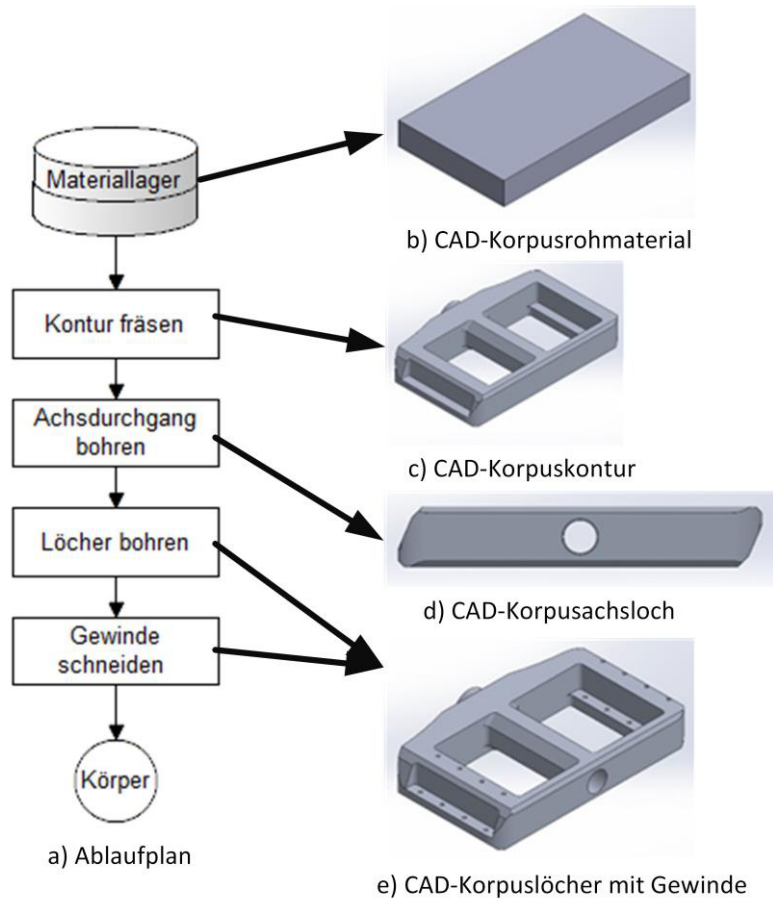


ABBILDUNG 59 CAD-CAM ASSETS

```
00:27:44: INFO: Werkzeugwechsel von T1001 zu T1002
00:27:55: INFO: Werkzeug T1120 eingewechselt.
00:28:02: INFO: Werkzeugwechsel von T1002 zu T1004
00:28:13: INFO: Werkzeug T5000 eingewechselt.
```

ABBILDUNG 60 BEISPIEL MASCHINENLOG

	Maschine	Werkstück	
X	-0.004	-0.004	mm
Y	-0.007	-0.007	mm
Z	-0.021	-0.021	mm
B	0.0000	0.0000	°
C	0.0000	0.0000	°
F		0 mm/min	
69%		0.000 mm/Umdr	
Spindeldrehzahl (max 42000):			
Soll:		0 Umdr/min	
Temp: 0°		Vibration:	
Vorschmierzyklus 00:00:00			

ABBILDUNG 61 BEISPIEL MASCHINENSCHNAPPSCHUSS

```

289 <SystemUnitClassLib Name="HerstellerXY:A-Team://Services">
290   <Version>1.0.0</Version>
291   <SystemUnitClass Name="Starte_NC_Programm">
292     <SupportedRoleClass
293       RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]/Service" />
294     <SystemUnitClass Name="Starte_NC_Programm">
295       <SupportedRoleClass
296         RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]/Service/Service Funktion" />
297       <SystemUnitClass Name="NC Aufruf">
298         <SupportedRoleClass
299           RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]
300           /Service/Service Funktion/Funktionsaufrufnachricht" />
301         <SystemUnitClass Name="Body">
302           <SupportedRoleClass
303             RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]
304             /Service/Service Funktion/Nachricht/Nachrichtenteil" />
305           <SystemUnitClass Name="Location">
306             <Attribute Name="Typ">
307               <Value>URI</Value>
308             </Attribute>
309             <Attribute Name="Wert" />
310             <SupportedRoleClass
311               RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]
312               /Service/Service Funktion/Nachricht/Nachrichtenteil/Nachrichtenfeld" />
313             </SystemUnitClass>
314           </SystemUnitClass>
315         </SystemUnitClass>
316       <SystemUnitClass Name="NC Resultat">
317         <SupportedRoleClass
318           RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]
319           /Service/Service Funktion/Funktionsresultatnachricht" />
320       <SystemUnitClass Name="Body">
321         <SupportedRoleClass
322           RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]
323           /Service/Service Funktion/Nachricht/Nachrichtenteil" />
324       <SystemUnitClass Name="Log">
325         <Attribute Name="Typ">
326           <Value>String</Value>
327         </Attribute>
328         <Attribute Name="Wert" />
329         <SupportedRoleClass
330           RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]
331           /Service/Service Funktion/Nachricht/Nachrichtenteil/Nachrichtenfeld" />
332         </SystemUnitClass>
333       <SystemUnitClass Name="Schnapschuss">
334         <Attribute Name="Typ">
335           <Value>String</Value>
336         </Attribute>
337         <Attribute Name="Wert" />
338         <SupportedRoleClass
339           RefRoleClassPath="[I4.0_Konzept:Base://RoleLib]
340           /Service/Service Funktion/Nachricht/Nachrichtenteil/Nachrichtenfeld" />
341         </SystemUnitClass>
342       </SystemUnitClass>
343     </SystemUnitClass>
344   </SystemUnitClass>
345 </SystemUnitClassLib>
346 </SystemUnitClassLib>

```

ABBILDUNG 62 EXERON NC STARTE SERVICE

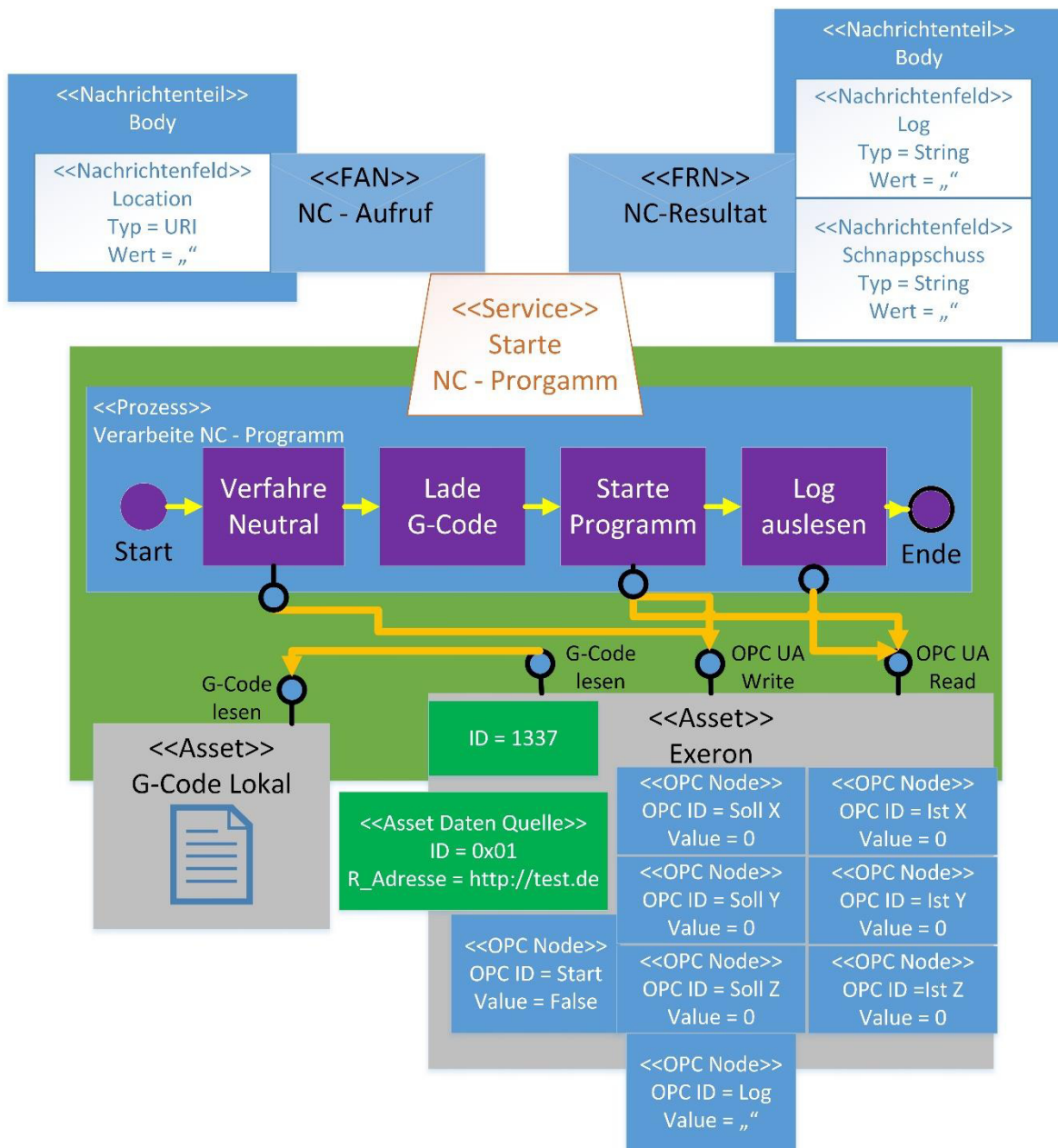


ABBILDUNG 63 EXERON VS

Für die Validierung ist es nötig, dass das umgesetzte Modell ein solches Beispiel abbilden kann. Es wird angenommen, dass der Hersteller der NC-Maschine für die Grundfunktionalität des Einlesens und Abarbeitens eines NC – Programmes eine I4.0-Komponente bereitstellt. Diese stellt einen Service zur Verfügung, welcher „Starte NC – Programm“ genannt wird. Dieser Service ist in *Abbildung 62* dargestellt und erhält über seine Funktionsaufrufnachricht eine URL im Nachrichtenfeld „Location“. Wie in *Abbildung 63* dargestellt, beginnt der Prozess „Verarbeite NC – Programm“ damit über das „OPC UA Write“ Interface auf die Maschine eine Soll-Position zu schreiben. Dies geschieht in der Aktivität „Verfahre Neutral“. Die durch einen „SequentialFlow“ verbundene Folgeaktivität „Lade G-Code“ nutzt die innerhalb der Nachricht übergebene URL. Die Aktivität downloadet von der übergebenen Adresse die G-Code Datei und instanziiert damit das Asset „G-Code Lokal“. Dieser Schritt könnte z.B. auch durch eine Subscription gelöst werden, in welcher die I4.0-Komponente „Exeron_VS“ so konfiguriert wird, dass diese das Asset von einer anderen I4.0-Komponente ausliest. Eine solche Konfiguration wurde weggelassen, um die Service Nutzung zu illustrieren. Die folgende Aktivität „Starte Programm“ schreibt

über das „OPC UA Write“ Interface auf den „Bool“-Knoten „Start“ den Wert „TRUE“. Dies führt dazu, dass die NC – Maschine über ihr „G – Code Lesen“ Interface den G – Code aus dem Asset „G – Code Lokal“ ausliest und dementsprechend dieses abarbeitet. Diese Aktivität liest vor dem Abschließen einen Schnappschuss der Maschine aus. Hierfür nutzt diese Aktivität einen gleichnamigen Unterprozess. Dieser liest mit der Aktivität „Schnappschuss lesen“ den Schnappschuss von den drei Koordinaten Knoten „X – Pos“, „Y – Pos“ und „Z – Pos“ aus und schreibt diese in die Resultatnachricht. Anschließend wartet die Aktivität „Warte auf Ende“ mit zyklischem Lesen auf den OPC UA Knoten „Start“, dass dieser den Wert „False“ annimmt und somit signalisiert, dass der Prozess am Ende ist. Die letzte Aktivität „Log auslesen“ stellt die finale Aktivität dar und liest die Log Ausgabe der Maschine aus. Sie wird im Folgenden genauer betrachtet. Abschließend wird der Prozess beendet und es wird die Resultatnachricht abgeschickt. Die *Abbildung 64* bietet einen Einblick in den Subprozess „Log auslesen“ von der Aktivität „Log auslesen“. Es wird mit der Aktivität „Lese Log“ das Log der Maschine ausgelesen und über einen „MessageFlow“ der ausgelesene Wert an den FieldAssigner „Schreibe Log Resultat“ weitergegeben. Der „MessageFlow“ referenziert hierbei die lokale Nachricht „TempNachricht“, welche die übertragene Nachricht widerspiegelt. Diese enthält die ausgelesenen Log - Informationen. Anschließend überträgt der FieldAssigner das Nachrichtenfeld „Log“ auf die Resultatnachricht und dessen Feld „Log“. Diese I4.0-Komponente wird mit der Instanziierung des Assets geliefert. Hier könnten z.B. Informationen über den Zusammenbau der Maschine enthalten sein. Auf solche Beschreibungen wurde für eine bessere Übersicht verzichtet. Eine Referenz auf das Handbuch gibt Aufschluss über die Informationsquelle, welche das Asset beschreibt.

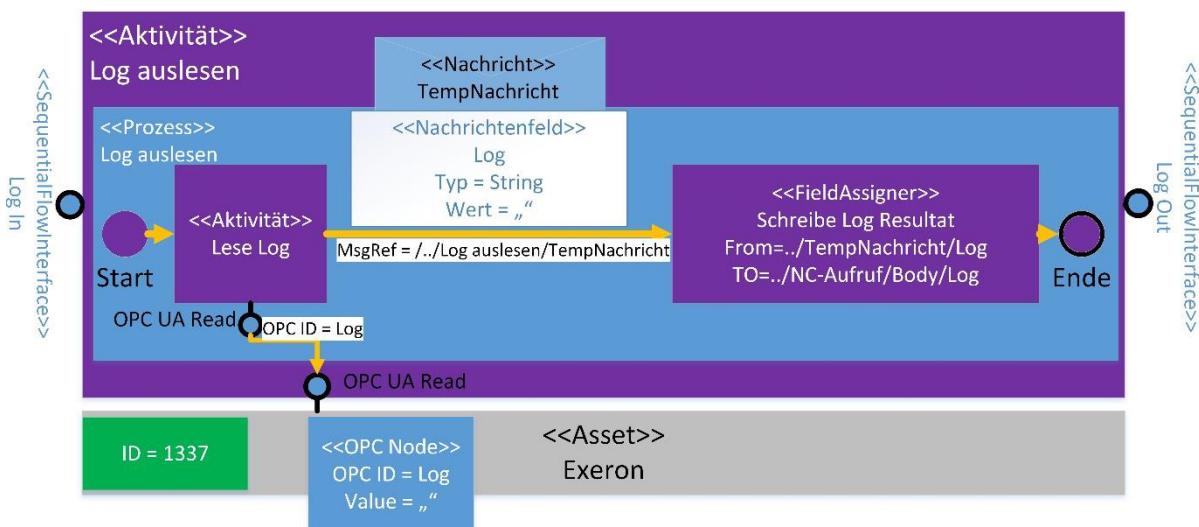


ABBILDUNG 64 SUBPROZESS LOG AUSLESEN

Die „Exeron VS“ wird von der Domäne „HerstellerXY:A-Team“ beschrieben und innerhalb der Domäne „Universität:ISW“ betrieben. Für die komplette Abbildung des Beispiels werden noch zwei weitere I4.0-Komponenten erstellt. Die Zusammenhänge dieser werden in der *Abbildung 65* dargestellt. Es handelt sich hierbei um die „CAM – Pedal VS“ und die „Korpus – Produktion VS“ I4.0-Komponenten. Die „CAM – Pedal VS“ dient der Abbildung für den Prozess, welcher die G – Code Dateien für die drei Fertigungsschritte erstellt. Diese verwaltet auch die hierbei erstellten Dateien und bildet sie als separates Asset ab. Der Service „CAM – Korpus“ beginnt einen stark vereinfachten Prozess, welcher aus einer Aktivität „Erstelle G – Code“ besteht. Dieser ist stark abstrahiert und verbindet die sechs Assets, welche für diesen Prozess nötig sind.

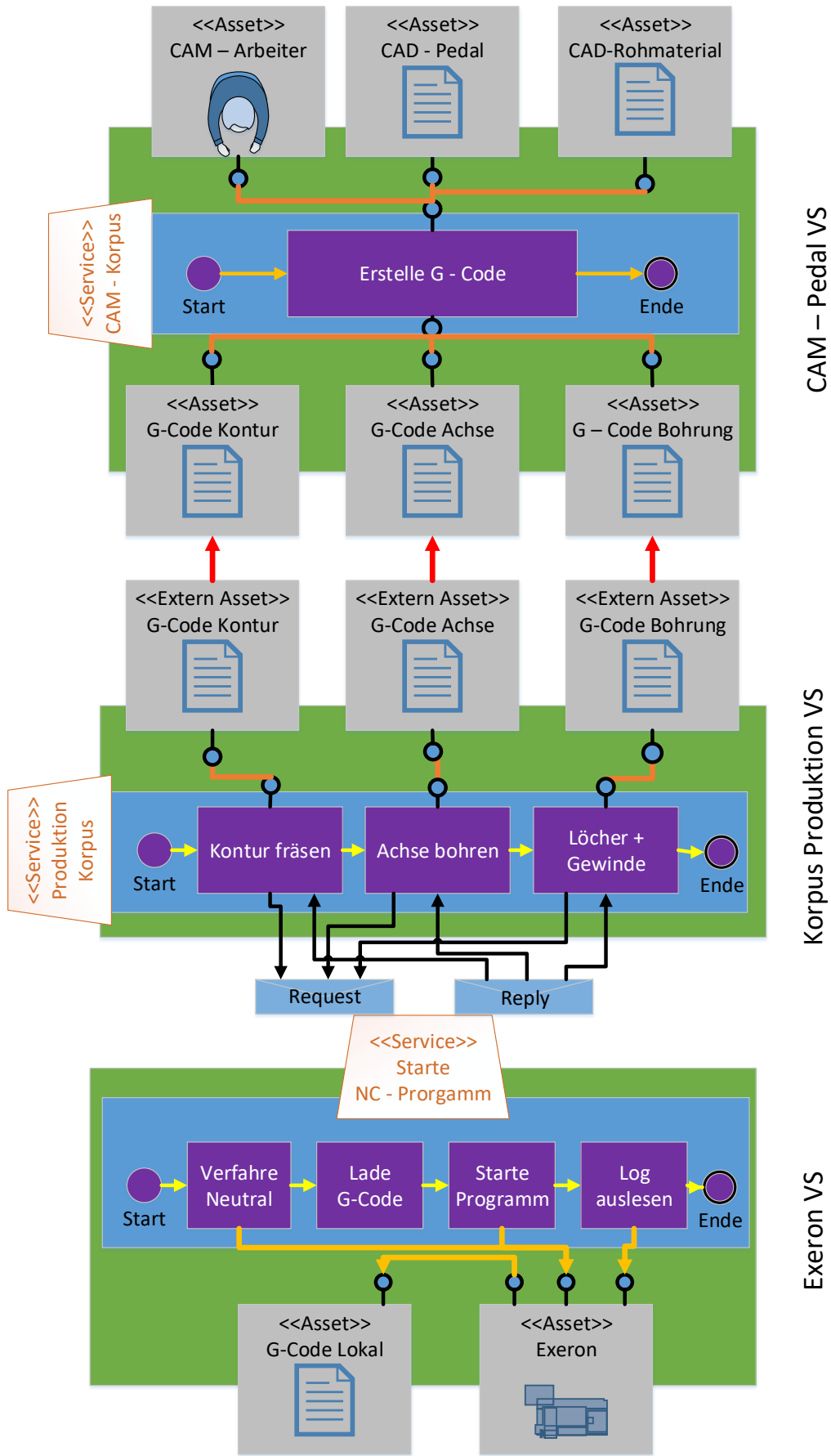


ABBILDUNG 65 VALIDIERUNGSBEISPIEL

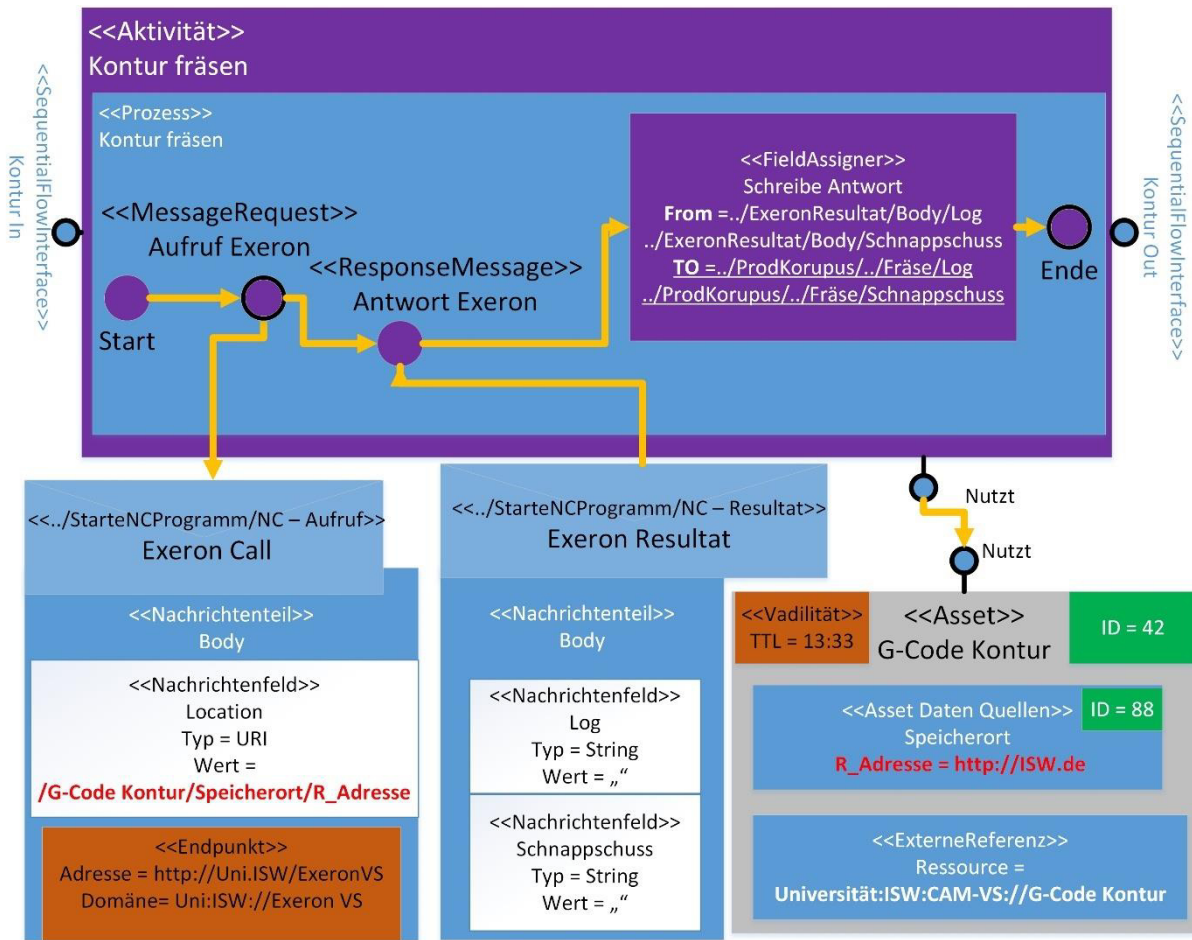


ABBILDUNG 66 AKTIVITÄT KONTUR FRÄSEN

Dies sind die drei Eingabe Assets „CAM – Arbeiter“, welches einen Mitarbeiter darstellt, „CAD – Pedal“, welches die CAD – Datei aus *Abbildung 58* abbildet und die „CAD – Datei“ für die Beschreibung des Rohmaterials. Das Resultat stellen die drei Assets „G-Code Kontur“, „G-Code Achse“ und „G-Code Bohrung“ dar, sie sind jeweils Assets vom Typ „G-Code“. Wird der Service aufgerufen, wird der „CAM-Arbeiter“ aus dem „CAD -Pedal“ und dem „CAD-Rohmaterial“ die drei G-Code Assets erstellen und wenn diese bereits existieren ersetzen. Die Hauptrolle dieser I4.0-Komponente ist in diesem Beispiel die drei resultierenden G – Code Assets bereit zu stellen. Diese werden anschließend von der I4.0-Komponente „Korpus Produktion VS“ als externes Assets eingebunden. Dies wird über eine Subscription dieser Elemente realisiert. Es wird hierbei ein Aktualisierungsintervall konfiguriert. Ein solches Intervall sorgt dafür, dass die Assets innerhalb der „Korpus Produktion VS“ ein QoP-Validität und eine „Externe Referenz“ erhalten. Zusätzlich werden bei einer erneuten Erstellung der G-Code Assets, diese automatisch in die I4.0-Komponente übertragen. Die I4.0-Komponente „Korpus Produktion VS“ ist für die Produktion des Korpus des Fahrradpedals zuständig. Sie besitzt den Service „Produktion Korpus“, welcher den Prozess für die Produktion beginnt. Hierfür sind die Aktivitäten „Kontur fräsen“, „Achse bohren“ und „Löcher + Gewinde“ sequentiell miteinander verbunden. Eine Aktivität nutzt eines ihrer zugehörigen G-Code Assets, um dessen Speicherort der NC – Maschine zu übergeben. Es wird im Folgenden die Aktivität „Kontur fräsen“ näher betrachtet. Diese wird in *Abbildung 66* dargestellt. Sie besteht aus einem gleichnamigen Prozess und wird initiiert durch das MessageRequest Event „Aufruf Exeron“.

Dieses nutzt die Nachricht „Exeron Call“, welche durch ihren Typ als Service Nachricht gekennzeichnet ist. Dieser Typ referenziert auf die Funktionsaufrufnachricht des Services „Starte NC Programm“ aus der Domäne „HerstellerXY:A-Team“. Die Nachricht wird mit einem Endpunkt versehen, um zu signalisieren, dass der Service auf der „Exeron VS“ innerhalb der Domäne „Universität:ISW“ aufgerufen wird. Der Wert für das Nachrichtenfeld „Location“ referenziert hierbei auf das „Asset Daten Quelle“ Element „Speicherort“ des Assets „G-Code Kontur“, um von diesem die „Ressourcen_Adresse“ auszulesen. Diese wurde innerhalb der Abbildung durch „R_Adresse“ abgekürzt und die beiden korrelierenden Werte rot markiert. Das folgende Event „Antwort Exeron“ ist ein Event vom Typ ResponseMessage und wartet auf die Ankunft der Antwortnachricht der „Exeron VS“. Diese Nachricht wird über das Element „Exeron Resultat“ abgebildet. Sie besitzt auch durch ihren Typen eine Referenz auf die FRN des Service. Durch die Ankunft dieser Nachricht folgt die finale Aktivität „Schreibe Antwort“, welche einen FieldAssigner darstellt. Dieser schreibt das Resultat dieses Aufrufes in die Resultatnachricht des umliegenden Service. Dies stellen alle benötigten Beschreibungen dar, welche für die Nachbildung der Beispielsituation benötigt werden. Abschließend wird diese I4.0-Komponente mit ihrer Beschreibung auf die „Universität:ISW“ Domänen-Registry veröffentlicht und registriert. Dies beinhaltet, dass auch unter dem Bereich „ServiceEndpunkte“ die Endpunkte für den erreichbaren Service und der I4.0-Komponente registriert werden. Für die Validierung wird abschließend der Aufruf des Service „Produktion Korpus“ betrachtet, welcher unter anderem die Aktivität „Kontur fräsen“ instanziiert. Er wird auf der Ressource „**Universität:ISW://Korpus Produktion VS Station** „ ausgeführt, auf dieser wird somit die I4.0-Komponente betrieben. Es wird sich hierbei auf die Ausführung dieser beschränkt und angenommen, dass die folgenden Aktivitäten simultan ablaufen. *Abbildung 67* bildet die Serviceinstanz-Ebene der laufenden I4.0-Komponente „Korpus Produktion VS“ ab. Der Service besitzt die FAN „Start Produktion“, welche eine leere Nachricht darstellt. Die Funktionsresultatnachricht „Ergebnis Produktion“ kapselt lediglich die Ergebnisse für die einzelnen Aufrufe der „Exeron VS“. Da der Fokus auf der Aktivität „Kontur fräsen“ liegt, wurde für eine bessere Übersicht die Felder für die restlichen Aktivitäten nicht dargestellt. Wie zu erkennen ist, besitzt jedes Element eine innerhalb der I4.0-Komponente „Korpus Produktion VS“ eindeutige ID, die in Kombination mit der URN der I4.0-Komponente global eindeutig ist. Ein wichtiger Punkt in der Abbildung ist die Tatsache, dass jedes Element nicht nur eine Instanz darstellt, sondern auch von einem anderen Typen dieser Instanz abgeleitet wird. So ist die Aktivität, welche in *Abbildung 66* dargestellt wurde, nicht mehr vom Typ „<<Aktivität>>“, sondern von dem verkürzt dargestellten Typ „<<./Kontur fräsen>>“. Die Abkürzung steht hierbei für die global eindeutige URN-Adresse „**Universität:ISW://Korpus Produktion VS Station/ Definition/Korpus Produktion VS/Korpus Produktion Prozess/Kontur fräsen**“, welche die Beschreibung der Aktivität innerhalb der Komponenten-Beschreibungsebene darstellt. Durch die Typenreferenz der Instanz ist die semantische Bedeutung der Instanz eindeutig, da diese die Instanz der Aktivität „Kontur fräsen“ darstellt. Nach diesem Schema werden alle Service- und Prozessbeschreibungen instanziiert. Dies gilt hierbei nicht für das Asset „G-Code Kontur“. Da dieses bereits eine Instanz innerhalb der Komponenten-Beschreibungsebene darstellt, wird diese in seinem Zustand in die Serviceinstanz-Ebene kopiert und mit einer neuen ID gekennzeichnet. Damit die Rolle dieser Instanz weiterhin eindeutig und auch dessen Herkunft definiert bleibt, wird diese mit dem zusätzlichen Attribut „RefID“ versehen. Dieses beinhaltet die ID „42“, welche die ID der kopierten Instanz darstellt. Für einen relativ zur Instanz-Ebene eindeutigen Namen muss dieser bei allen Elementen umbenannt werden. Dies resultiert daraus, dass durch ein wiederholtes Aufrufen des Service weitere Instanzen von diesem und damit verbundener Prozesselemente und Assets erstellt werden.

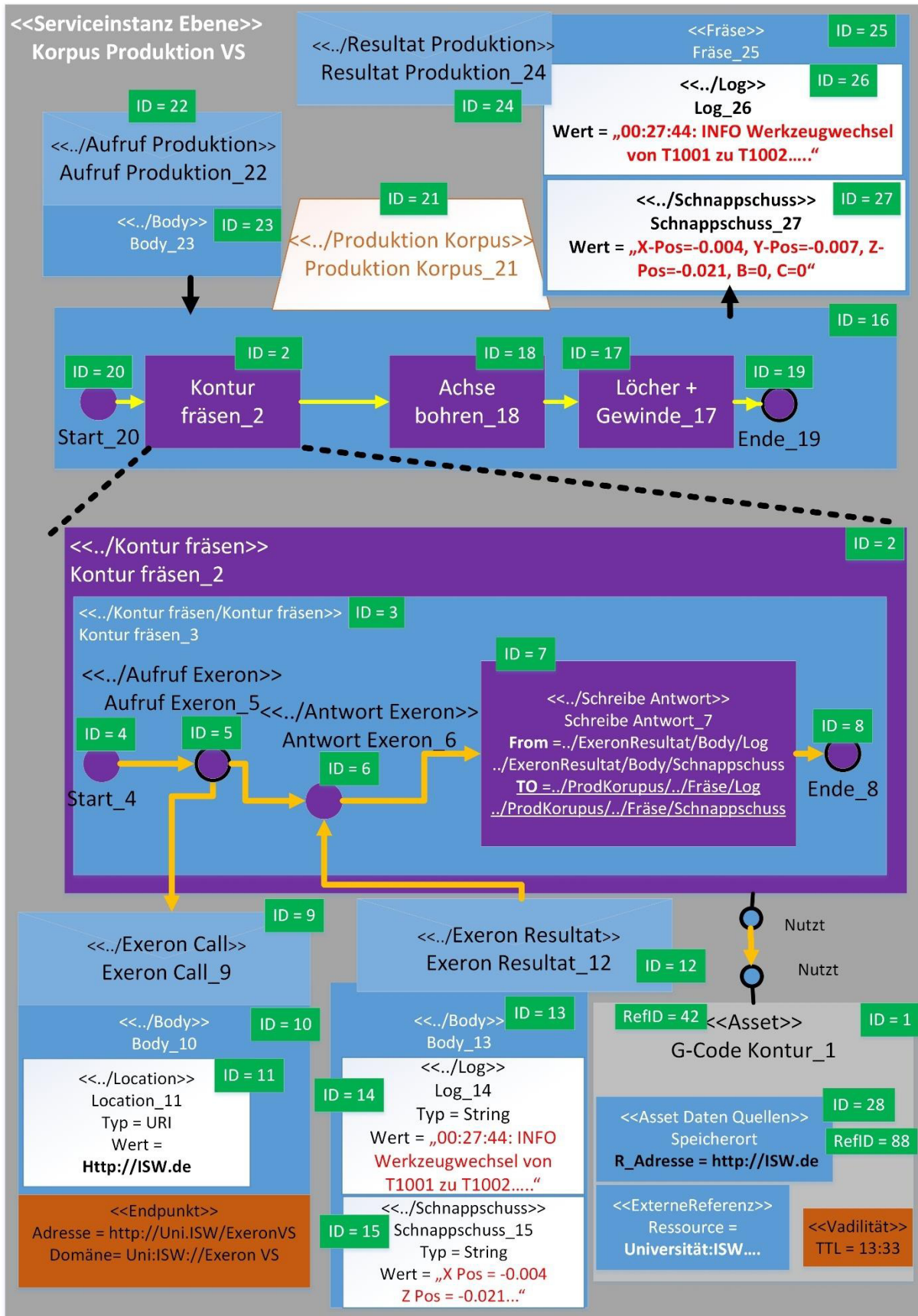


ABBILDUNG 67 SERVICEINSTANZ KORPUS 14.0-KOMPONENTE

Der Name stellt in der Instanz-Ebene aus diesem Grund keine Quelle für semantische Informationen dar. Diese Information befindet sich in der Typreferenz oder bei kopierten Instanzen innerhalb der „RefID“. In diesem Beispiel wurden die Ursprungsnamen mit ihrer ID erweitert, um eine Eindeutigkeit zu garantieren. Dies stellt keine Richtlinie dar, verbessert jedoch die Übersicht. Die Abbildung zeigt zudem die beiden angestrebten Resultate, die während der realen Produktion gesammelt wurden. Diese wurden rot markiert und treten innerhalb der FRN „Resultat Produktion_24“ auf. Es handelt sich hierbei um das ausgelesene Log der Maschine und einen willkürlichen Schnappschuss des Maschinenzustandes. Sie wurden, wie in der Beschreibung der Aktivität „Kontur fräsen“ definiert, aus der FRN „Exeron Resultat“ kopiert. Die Instanz dieser FRN wurde „Exeron Resultat_12“ genannt und mit der ID „12“ versehen. Dessen Werte, für den Schnappschuss und den Maschinen – Log, wurden über die Instanz der „Schreibe Antwort“ FieldAssigner-Aktivität übertragen. Diese Instanz wurde „Schreibe Antwort_7“ genannt, da diese analog die ID „7“ erhalten hat. Die dargestellten Elemente für den **Endpunkt**, die **externe Referenz** sowie die **Validität** besitzen keine ID, da diese dem umliegenden Element als AutomationML Rolle zugewiesen wurden. Dies bedeutet, dass es sich hierbei um keine eigenständigen Elemente handelt, sondern diese ihre Attribute an das zugewiesene Element binden. Dieses Beispiel illustriert einen Ausschnitt aus der Entwicklung einer I4.0-Komponente und die Ausführung eines ihrer Service, um an die angestrebten Informationen zu gelangen.

4.5 VALIDIERUNG DER ANFORDERUNGEN

Es werden im Folgenden die einzelnen Anforderungen an das Datenmodell aus *Kapitel 3.1* aufgelistet. Diese Auflistung enthält jeweils eine Begründung, wie durch das erstellte Datenmodell und dessen Umsetzung, mithilfe von AutomationML, die aufgestellten Anforderungen erfüllt werden.

Plattformunabhängigkeit: Diese wird, wie in *Kapitel 4.1* bereits erläutert, durch das auf XML basierende AutomationML Format erzielt. Durch die Nutzung von XML ist dieses auf verschiedenen Plattformen einlesbar und es stehen eine Vielzahl an Werkzeugen bereit, um auf der jeweiligen Plattform damit zu arbeiten. Für die Beispiele innerhalb dieser Arbeit wurde z.B. der AutomationML Editor und Microsofts Code genutzt.

SOA: Die Vorgabe ein auf Service ausgerichtetes Modell zu entwerfen hat dafür gesorgt, dass die Datenhaltung analog eine SOA voraussetzt. So wurde die Domänen – Registry eingeführt, welche innerhalb des SOA Dreiecks die Rolle der Registry übernimmt. Diese wurde zusätzlich erweitert, da hierbei nicht nur Servicebeschreibungen, sondern auch I4.0-Komponenten- und Asset-Beschreibungen verwaltet werden. Zudem kann diese Domänen-Registry bezüglich laufender Instanzen eines Service oder einer I4.0 Komponente auf einem Endpunkt durchsucht werden. Wird eine I4.0-Komponente oder ein Service gefunden, ist es möglich diese zu nutzen. Dies entspricht der Referenz auf abstrakte Beschreibungen eines Service oder einer I4.0-Komponente innerhalb einer Servicenutzung oder I4.0-Komponentenbeschreibung. Für die Vollständigkeit ist es möglich auf dieser Domänen-Registry neue Services und I4.0-Komponentenbeschreibungen zu veröffentlichen.

Lose Kopplung: Die lose Kopplung setzt sich aus den vier Autonomien Referenz-, Plattform-, Format- und Zeitautonomie zusammen. Die Plattformautonomie wird durch die Plattformunabhängigkeit und somit der Verwendung von XML erfüllt. Die Formatautonomie wird ebenfalls über die Verwendung von XML als Übertragungsformat erfüllt. Dieses erlaubt es, dass verschiedene Endpunkte voneinander abweichende Datenformate über XML normalisieren und transferieren können. So kann ein Service, welcher in seiner Ausführung ADS Daten verarbeitet, mit einem Service kommunizieren, welcher in seiner Ausführung OPC UA Daten verarbeitet. Durch die Verwendung von Nachrichten kann die

Zeitautonomie erfüllt werden, da diese keine direkte Bindung an einen physikalischen Endpunkt besitzt und somit von der Zeit unabhängig einem dafür vorgesehenen logischen Endpunkt zugewiesen werden kann. Wird es verlangt, dass diese Autonomie erfüllt wird, wird somit auch für die Übertragung der Nachrichten eine Form von MQM vorausgesetzt, die gewährleistet, dass keine Nachricht verloren geht. Die letzte Bedingung ist die Referenzautonomie und wird durch die Verwendung keiner physikalischen Adressen für Endpunkte oder I4.0-Komponentenbeschreibungen erfüllt. Es handelt sich hierbei stets um einen logischen Endpunkt oder eine logische Beschreibung, welche über eine URN referenziert wird. Dies erlaubt es die URL eines Endpunktes zu verändern, sollte dieser auf ein anderes physikalisches System transferiert und dort weiter betrieben werden.

Wiederverwertbarkeit: Das resultierende Modell ist dafür ausgelegt, dass jeder von einer I4.0-Komponente genutzter Service ausgetauscht werden kann, indem man den Endpunkt auf welchen die Nachrichten verschickt werden analog anpasst. Dies macht die Nutzung einer I4.0-Komponente austauschbar, da sie durch eine andere I4.0-Komponente ersetzt werden kann, welche dieselben Services bietet. Es ist möglich diese durch mehrere I4.0-Komponenten zu ersetzen, welche in der Summe dieselben Services bieten und semantisch eine äquivalente Aufgabe verrichten. Zudem ist es möglich durch die Beschreibung von Assets innerhalb der abstrakten Beschreibungsebene semantisch äquivalente Asset-Instanzen auszutauschen. So kann innerhalb des Validierungsbeispiels die „Exeron“ Maschine durch eine andere Instanz der „Exeron“ Maschine ersetzt werden, da diese durch eine gemeinsame abstrakte Beschreibung typgleich sind. Es ist möglich diese Beschreibungen innerhalb dieser Ebene soweit abstrakt zu halten, dass dadurch theoretisch jede Art von NC-Maschinen untereinander austauschbar werden.

Keine Technologieabhängigkeiten: Das Datenmodell lehnt sich an die Verwendung von Web Services, BPMN und BPEL an. Es wurde darauf verzichtet, deren XML-Repräsentation für die Verwendung anzupassen und zu nutzen, um keine Abhängigkeiten zu der Technologie Web Services zu erhalten. Das Modell hält jegliche Form von Definition abstrakt, da zum Zeitpunkt dieser Arbeit nicht klar ist welche Technologien genutzt werden sollen. Diese können dann auf die jeweilige Technologie übertragen werden, sobald diese festgelegt sind.

I4.0-Komponentendatenmodell: Die einzelnen Plattformservices für die Verwaltung von I4.0-Komponenten wurden innerhalb von *Kapitel 4.2* definiert und ergänzt. Durch die Verwendung von AutomationML als Metamodell ist es möglich jegliche Form von Assets innerhalb der Wertschöpfungskette abzubilden und zu nutzen [32]. Abschließend erfüllt das Modell die Eigenschaften für eine Verwaltungsschale. Es ist möglich Dienste und Zustände zu definieren, welche über die Modellierung von Services und Prozessen mit Start-, End- und Zwischenevents stattfinden. Diese können auch innerhalb eines Assets über eine proprietäre Definition geschehen. Zudem ist es möglich QoS und Sicherheitseigenschaften eines Service darzustellen.

5 ZUSAMMENFASSUNG UND AUSBLICK

In dieser Arbeit wurde zunächst über den Stand der Technik und die Grundlagen eine Wissensbasis geschaffen. Diese sammelte Informationen über verschiedene Bereiche, welche innerhalb der Digitalen Produktion aufeinandertreffen und miteinander agieren. Es wurde hierbei ein Fokus auf die Modelle hinter Technologien wie NC, PLC, OPC oder ADS gelegt. Zusätzlich wurden neben diesen konkreten Technologien auch abstrakte Modelle wie UML, BPMN, AutomationML oder Teile des CAX Bereichs betrachtet. Diese lieferten Informationen über die Darstellung von Assets durch die Modellierung mithilfe von AutomationML sowie die Darstellung von Prozessen durch BPMN, welche von einem Serviceaufruf gestartet werden. Diese Wissensbasis wurde abgeschlossen mit den Informationen über den aktuellen Stand der Entwicklungen des VDE GMA Fachausschuss „Industrie 4.0“, welcher unter anderem durch RAMI 4.0 die Rahmenbedingungen für diese Arbeit definiert. Diese Informationen wurden genutzt, um die Anforderungen an das Datenmodell und dessen Datenhaltung zu bestimmen und in *Kapitel 3.1* festgehalten. Anschließend wurde die Entwicklung eines imaginären Schrankes als Produkt analysiert.

Es wurden hierbei, unter Berücksichtigung der Wertschöpfungsketten, einzelne Schritte einer solchen Produktion in einer I4.0-Komponente betrachtet. Diese Betrachtung der einzelnen Schritte legte in *Kapitel 3.2* einzelne Entitäten und ihre Abhängigkeiten zueinander für das Modell offen. Zudem wurde der Bedarf für eine Domänen-Registry ermittelt, welche die Rolle der Registry innerhalb des SOA – Dreiecks einnimmt und zusätzlich abstrakte Beschreibungen von Assets und I4.0-Komponenten verwaltet. Diese Analyse führte zu einem abstrakten Datenmodell, welches sich an bereits bestehenden Modellen wie BPMN, BPEL, Web Services und AutomationML anlehnt. Das Verwenden dieser verschiedenen Modelle innerhalb eines Datenmodells ermöglicht es alle Anforderungen zu erfüllen. Es ist möglich mit der Anlehnung an AutomationML jegliche Assets zu beschreiben [32] und zwischen einer Beschreibung und der dazugehörigen Instanz zu differenzieren. Die Modelle hinter Web Services, BPEL und BPMN wurden genutzt, um wohldefinierte Services sowie Prozesse zu beschreiben sowie zu definieren wie diese Prozesse instanziiert werden. Zusätzlich wurden diese erweitert, um z.B. die Abhängigkeit eines Prozesses zu einem Asset modellieren zu können und somit den Übergang von einem neutralen Datenformat zur proprietären Asset-Schnittstelle zu ermöglichen. Dies führte zu dem in *Kapitel 3.3* vorgestellten Datenmodell. Dieses Datenmodell wurde anschließend durch AutomationML in eine maschinenlesbare Form umgesetzt. Es wurde eine an die Technologie angepasste Datenhaltung innerhalb von *Kapitel 4.2* erörtert. Die Datenhaltung führt die drei verschiedenen Ebenen **abstrakte Beschreibungsebene**, **Komponenten-Beschreibungsebene** und **Serviceinstanz-Ebene** der Datenhaltung ein. Diese Ebenen stehen wie in *Abbildung 68* mit der Definition des SOA – Dreiecks in Beziehung. Für die Nutzung einer I4.0-Komponente muss hierfür deren abstrakte Beschreibung auf einer Domänen-Registry veröffentlicht werden. Diese Registry liegt in der Zuständigkeit des Komponentenerstellers. Dieser veröffentlicht die Beschreibung von definierten Assets, Services, Interfaces und deren Zusammenhänge innerhalb der I4.0-Komponente. Zusätzlich werden Referenzen auf importierte Elemente gesetzt, sollte z.B. ein Asset oder die Servicedefinition eines anderen Herstellers innerhalb der Komponente genutzt werden. Der Betreiber einer I4.0-Komponente veröffentlicht die Service-Endpunkte, über welche eine I4.0-Komponente oder Services angefragt werden können. Der Nutzer eines Service oder einer I4.0-Komponente kann auf der jeweiligen Domänen-Registry nach einem benötigten Service oder I4.0-Komponente suchen und über die Referenz des Endpunktes diese nutzen. Hierfür kann der Nutzer über die Plattformdienste mit einer I4.0-Komponente interagieren und dessen Zustand auslesen. Der Nutzer ist somit auch in der Lage die Services aufzurufen, welche von einer I4.0-Komponente bereitgestellt werden. Anknüpfend an die

Definition der Datenhaltung durch AutomationML wurde das abstrakte Datenmodell in AutomationML realisiert und anschließend das Datenmodell über die Modellierung eines Beispiels illustriert sowie validiert. Abschließend wurde geprüft ob das Datenmodell alle Anforderungen erfüllt.

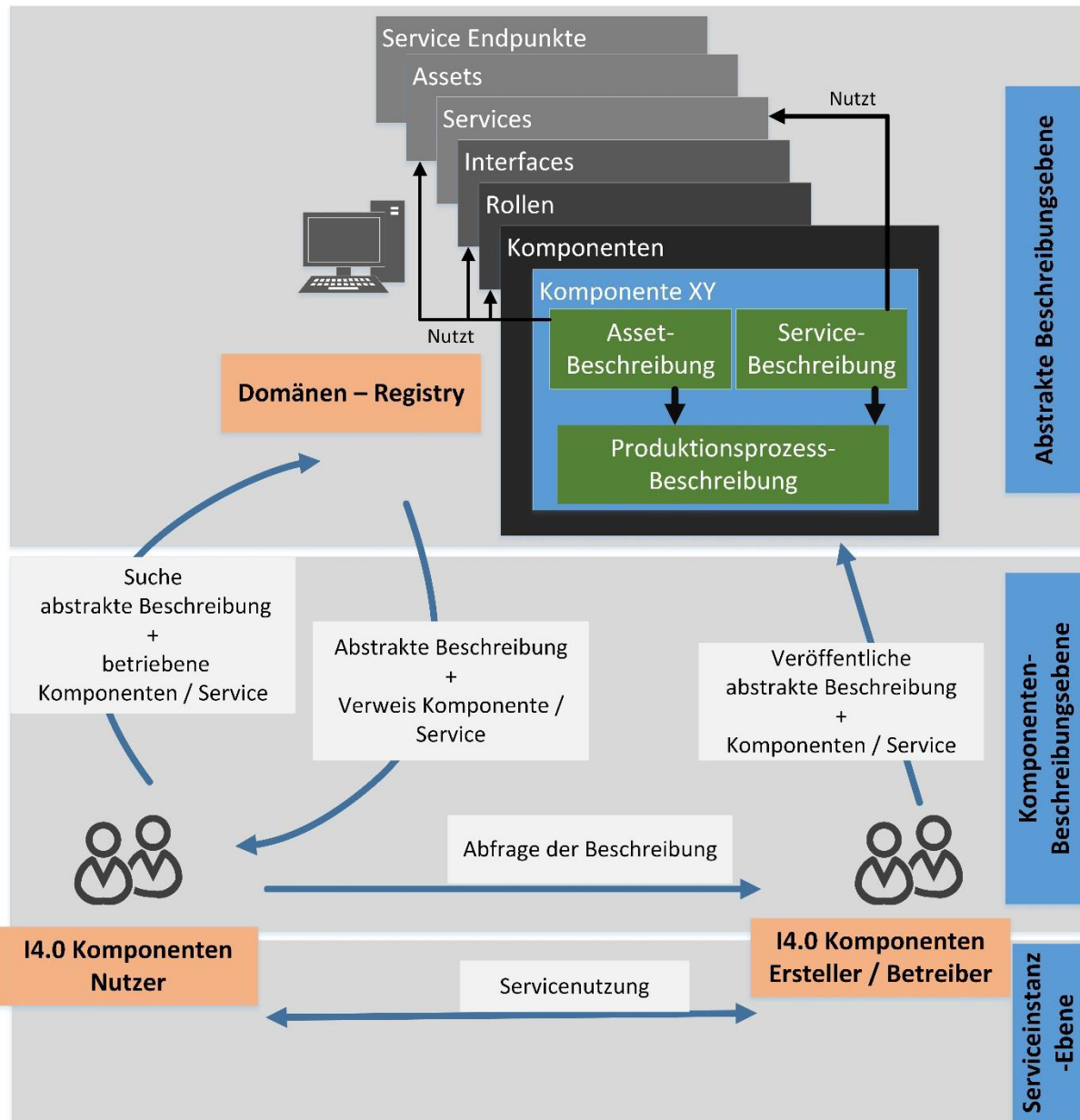


ABBILDUNG 68 SOA BESCHREIBUNGSEBENEN

Das Resultat dieser Arbeit ist ein mehrschichtiges Modell, welches als Metamodell, die Modelle von AutomationML und CAEX nutzt. Aufbauend auf diesen wurde ein an AutomationML, Web Services, BPMN und BPEL angelehntes Datenmodell als Metamodell erstellt, um darin Assets, Services, Prozesse und I4.0-Komponenten zu beschreiben. Dieses Metamodell kann genutzt werden, um eine I4.0 konforme Komponentenbeschreibung zu erstellen und definiert wie diese veröffentlicht sowie von dritten genutzt werden kann. Die letzte Modellebene stellt die Instanziierung von Services und Assets nach ihrer Beschreibung innerhalb des Modells dar. *Abbildung 69* illustriert diesen Aufbau. Es handelt sich dennoch um ein Datenhaltungskonzept, da noch nicht alle Elemente wohldefiniert sind. So befindet sich die Definition was eine I4.0 konforme Kommunikation ist noch in der Planung und die vollständige Validierung, ob dieses Datenmodell in allen Anwendungsszenarien von Industrie 4.0 ausreichend ist,

muss noch erfolgen. Das hier präsentierte Modell dient als Idee für die Basis einer konkreten Umsetzung sowie für die Entwicklung zukünftiger Standards. Dieses Modell bietet durch die Einführung der abstrakten Beschreibungsebene, in welcher Services, Assets, Rollen und Interfaces ohne konkrete Instanz definiert werden können, die Möglichkeit Beschreibungen zu generalisieren. Diese Generalisierungen können als Standarddefinition für Assets oder Services festgelegt werden und bieten somit Raum für die Vereinheitlichung des angestrebten I4.0 Ökosystems, ohne diese einzelnen Herstellern aufzuzwingen. Es bleibt abzuwarten inwieweit dieses Datenhaltungskonzept den noch folgenden Anforderungen an Industrie 4.0 genügt und somit das Potential besitzt, für das Fundament der kommenden industriellen Revolution einen Beitrag zu leisten.



ABBILDUNG 69 MODELLIERUNGSEBENEN

6 LITERATURVERZEICHNIS

- [1] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, „Industrie 4.0 Wertschöpfungsketten“, 2014. [Online]. Available: https://www.vdi.de/fileadmin/vdi_de/redakteur_dateien/sk_dateien/VDI_Industrie_4.0_Wertschoepfungsketten_2014.pdf. [Zugriff am 22 8 2016].
- [2] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, „Industrie 4.0 Statusreport Gegenstände, Entitäten, Komponenten“, 4 2014. [Online]. Available: https://www.vdi.de/fileadmin/vdi_de/redakteur_dateien/sk_dateien/VDI_Industrie_4.0_Komponenten_2014.pdf. [Zugriff am 21 9 2016].
- [3] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, „Referenzarchitekturmodell Industrie 4.0“, 4 2015. [Online]. Available: https://www.vdi.de/fileadmin/user_upload/VDI-GMA_Statusreport_Referenzarchitekturmodell-Industrie40.pdf. [Zugriff am 24 8 2016].
- [4] J. Wolf, *Steuerungsintegrierte, adaptive Programmausführung einer aufgabenorientierten Programmierung in STEP-NC*, Lehrstuhl für Werkzeugmaschinen, 2009.
- [5] G. Pritschow, *Einführung in die Steuerungstechnik*, Hanser München, 2006.
- [6] H.-J. a. G. U. Gevatter, *Handbuch der Mess- und Automatisierungstechnik in der Produktion*, Springer-Verlag, 2013.
- [7] W. a. L. S.-H. a. D. M. Mahnke, *OPC unified Architecture*, Springer Science Business Media, 2009.
- [8] OPC Foundation, „Interoperabilität für Industrie 4.0 und das Internet der Dinge“, 4 2015. [Online]. Available: <https://opcfoundation.org/wp-content/uploads/2015/04/OPC-UA-Interoperability-For-Industrie4-and-IoT-DE1.pdf>. [Zugriff am 19 9 2016].
- [9] commsvr, „OPC UA Modell“, 2010. [Online]. Available: <http://www.commsvr.com/uamodeldesigner/html/7cd6ebfb-3582-4ae7-8b1c-89e038d9bb66.htm>. [Zugriff am 22 8 2016].
- [10] Beckhoff, „Beckhoff“, 18 8 2016. [Online]. Available: http://infosys.beckhoff.com/index.php?content=../content/1031/tcadswebservice/html/webservice_intro.htm&id=.
- [11] Beckhoff, „Beckhoff TwinCAT“, [Online]. Available: http://infosys.beckhoff.com/index.php?content=../content/1031/tcadswebservice/html/webservice_intro.htm&id=. [Zugriff am 18 8 2016].
- [12] Beckhoff, „Beckhoff ADS Fieldbus“, [Online]. Available: http://www.beckhoffautomation.com/usa/pdf/press/Types_of_Fieldbus.pdf. [Zugriff am 18 8 2016].

- [13] *Object Management Group*, „UML 2.5,“ 3 2015. [Online]. Available: <http://www.omg.org/spec/UML/2.5/>. [Zugriff am 23 8 2016].
- [14] *OMG*, „Meta Object Facility,“ [Online]. Available: <http://www.omg.org/mof/>. [Zugriff am 23 8 2016].
- [15] *Object Management Group*, „BPMN 2.0,“ [Online]. Available: <http://www.omg.org/spec/BPMN/2.0/>. [Zugriff am 23 8 2016].
- [16] *G. a. K. F.-L. Spur*, „Das virtuelle produkt,“ München: Hanser, 1997.
- [17] *G. Pahl*, *Konstruieren mit 3D-CAD-Systemen: Grundlagen, Arbeitstechnik, Anwendungen*, Springer-Verlag, 2013.
- [18] *Dassault Systèmes*, „3ds Dassault Systèmes CATIA,“ [Online]. Available: <http://www.3ds.com/>. [Zugriff am 24 8 2016].
- [19] *I. Zeid*, *CAD/CAM theory and practice*, McGraw-Hill Higher Education, 1991.
- [20] *AutomationML Initiative*, „AutomationML Whitepaper Part 1 - Architecture and general requirements,“ 11 4 2016. [Online]. Available: https://www.automationml.org/o.red/uploads/dateien/1460366687-AutomationML%20Whitepaper%20Part%201%20-%20AutomationML%20Architecture%20v2_2016Apr.pdf. [Zugriff am 19 9 2016].
- [21] *AutomationML Initiative*, „AutomationML Whitepaper Part 2 - Role class libraries,“ 04 12 2014. [Online]. Available: https://www.automationml.org/o.red/uploads/dateien/1417686992-AutomationML%20Whitepaper%20Part%202%20-%20AutomationML%20Role%20Class%20Libraries%20v2_Oct2014.pdf. [Zugriff am 19 9 2016].
- [22] *AutomationML Initiative*, „AutomationML Whitepaper Part 3 - Geometry and Kinematics,“ 5 11 2015. [Online]. Available: https://www.automationml.org/o.red/uploads/dateien/1446724412-AutomationML%20Whitepaper%20Part%203%20-%20AutomationML%20Geometry%20v2_Aug2015.pdf. [Zugriff am 19 9 2016].
- [23] *AutomationML Initiative*, „AutomationML Whitepaper Part 4 - Logic Description,“ 4 12 2014. [Online]. Available: https://www.automationml.org/o.red/uploads/dateien/1417686916-AutomationML%20Whitepaper%20Part%204%20-%20AutomationML%20Logic%20Description%20v2_Aug2013.pdf. [Zugriff am 12 9 2016].
- [24] *AutomationML Initiative*, „AutomationML BPR external Referenzen,“ 08 08 2016. [Online]. Available: <https://www.automationml.org/o.red.c/news-182.html>. [Zugriff am 14 09 2016].
- [25] *VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik*, „Industrie 4.0 Statusreport Auf dem Weg zu einem Referenzmodell,“ 2014. [Online]. Available:

https://www.vdi.de/fileadmin/vdi_de/redakteur_dateien/sk_dateien/VDI_Industrie_4.0_Referenzmodell_2014.pdf. [Zugriff am 19 9 2016].

- [26] CEN-CENELEC-ETSI Smart Grid Coordination Group, „SGAM,“ 11 2012. [Online]. Available: http://ec.europa.eu/energy/sites/ener/files/documents/xpert_group1_reference_architecture.pdf. [Zugriff am 25 8 2016].
- [27] T. Erl, *Service-oriented architecture: a field guide to integrating XML and web services*, Prentice Hall PTR, 2004.
- [28] C. a. F. J. Ferris, „What are web services?,“ *Communications of the ACM*, Bd. 6, Nr. 46, p. 31, 2003.
- [29] F. a. D. M. a. K. R. a. N. W. a. M. N. a. W. S. Curbera, „Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI,“ *IEEE Computer Society*, Bd. 2, Nr. 6, p. 86, 2002.
- [30] J. a. S. A. a. M. J. a. A. J. a. K. K. Cardoso, „Quality of service for workflows and web service processes,“ *Web Semantics: Science, Services and Agents on the World Wide Web*, Bd. 1, Nr. 3, pp. 281--308, 2004.
- [31] AutomationML Initiative, „AutomationML Constraints,“ 10 2014. [Online]. Available: https://www.automationml.org/o.red/uploads/dateien/1417688745-BPR_001E_Constraint_RegEx_Oct2014.pdf. [Zugriff am 14 09 2016].
- [32] a. P. D.-I. h. A. L. L. H. N. S. M. S. D.-I. R. D. J. P. O. G. D.-I. M. S. M. S. S. M. D.-I. E. P.-B. Anton Hirzle, „sps-magazin AutomationML Fachexperten erklären das Format,“ 2014. [Online]. Available: <http://www.sps-magazin.de/downloads/WhitepaperAutomationML.pdf>. [Zugriff am 14 9 2016].
- [33] Siemens, „plm Automation Siemens NX CAM,“ [Online]. Available: [https://www.plm.automation.siemens.com/de_ch/products/nx/for-manufacturing/cam/3-axis-milling.shtml#lightview%26url=/de_ch/Images/Siemens-PLM-NX-CAM-fs_tcm782-64484.pdf%26title=NX CAM%26description=NX CAM Fact Sheet%26docType=pdf](https://www.plm.automation.siemens.com/de_ch/products/nx/for-manufacturing/cam/3-axis-milling.shtml#lightview%26url=/de_ch/Images/Siemens-PLM-NX-CAM-fs_tcm782-64484.pdf%26title=NX%26description=NX%26docType=pdf). [Zugriff am 24 8 2016].
- [34] webme.com, „KOP Plan,“ [Online]. Available: <http://img.webme.com/pic/s/sps-anlagen/kop-xor.png>. [Zugriff am 19 8 2016].

7 ABBILDUNGSVERZEICHNIS

Abbildung 1 NC Architektur [5]	3
Abbildung 2 PLC Zyklus.....	4
Abbildung 3 Kontaktplan [34] (links) und Anweisungsliste (rechts).....	5
Abbildung 4 G-Code Beispiel.....	5
Abbildung 5 OPC Data Access [7]	6
Abbildung 6 OPC UA Kommunikation [8].....	8
Abbildung 7 OPC UA Metamodel [9].....	8
Abbildung 8 ADS Systemarchitektur [12].....	9
Abbildung 9 UML Metamodellhierarchie.....	11
Abbildung 10 Entity-Relationship.....	11
Abbildung 11 Flussdiagramm in UML.....	12
Abbildung 12 BPMN Pools und Lanes	13
Abbildung 13 BPMN Activity.....	13
Abbildung 14 BPMN Flow	14
Abbildung 15 BPMN Events.....	14
Abbildung 16 BPMN Gateway.....	15
Abbildung 17 CAD CATIA Beispiel [18]	16
Abbildung 18 NX Siemens [33].....	17
Abbildung 19 AutomationML InterfaceClassLibrary	18
Abbildung 20 AutomationML RoleClassLibrary.....	19
Abbildung 21 AutomationML SystemClassLibrary.....	19
Abbildung 22 AutomationML InstanceHierarchy.....	20
Abbildung 23 AutomationML Editor.....	21
Abbildung 24 Industrie 4.0 Wertschöpfungsketten [1]	21
Abbildung 25 Wertschöpfungskette Produkt- und Produktlinienentwicklung [1].....	22
Abbildung 26 Wertschöpfungskette Verfahrens- und Anlagenentwicklung [1]	22
Abbildung 27 Wertschöpfungskette Produktproduktion und After Sales Services [1].....	23
Abbildung 28 Wertschöpfungskette Technische Anlage [1].....	23
Abbildung 29 RAMI4.0 Architektur [3].....	25
Abbildung 30 Lebenszyklus [3].....	26
Abbildung 31 I4.0-Komponente [3]	26
Abbildung 32 Verwaltungsschale-Datenformat [3].....	27
Abbildung 33 Verwaltungsschalenvernetzung [3]	28
Abbildung 34 Servicearchitekturmodell	30
Abbildung 35 Paxx-Schrank CAD-Skizze	33
Abbildung 36 Paxx - Kleiderschrank Abstraktion	34
Abbildung 37 Externe Asset Abhängigkeit.....	35
Abbildung 38 Teilprozess-R-Seite-Paxx.....	36
Abbildung 39 Teilprozess mit I4.0-Komponenten	37
Abbildung 40 Subprozess Bohrung - Span - R	38
Abbildung 41 Bohrungsprozess.....	39
Abbildung 42 I4.0-Domäne Datenmodell	41
Abbildung 43 Servicebeschreibung Datenmodell	42
Abbildung 44 Prozessbeschreibung Datenmodell	45

Abbildung 45 I4.0 Asset Datenmodell	47
Abbildung 46 I4.0-Komponente Datenmodell	50
Abbildung 47 Gesamtes Datenmodell	51
Abbildung 48 Domänen-Registry Beschreibung.....	53
Abbildung 49 Beschreibungsebenen	54
Abbildung 50 Initiale Komponente	56
Abbildung 51 Beispiel Servicebeschreibung mit Endpunkt.....	59
Abbildung 52 Servicebeschreibung AutomationML.....	61
Abbildung 53 Prozessbeschreibung AutomationML.....	62
Abbildung 54 Externe Referenz.....	63
Abbildung 55 Quality of Presentation	64
Abbildung 56 Asset Beschreibungselemente	64
Abbildung 57 I4.0-Komponente AutomationML.....	65
Abbildung 58 Fahrradpedal CAD.....	66
Abbildung 59 CAD-CAM Assets	67
Abbildung 60 Beispiel Maschinenlog.....	67
Abbildung 61 Beispiel Maschinenschnappschuss.....	67
Abbildung 62 Exeron NC Starte Service	68
Abbildung 63 Exeron VS.....	69
Abbildung 64 Subprozess Log auslesen	70
Abbildung 65 Validierungsbeispiel	71
Abbildung 66 Aktivität Kontur fräsen	72
Abbildung 67 Serviceinstanz Korpus I4.0-Komponente	74
Abbildung 68 SOA Beschreibungsebenen.....	78
Abbildung 69 Modellierungsebenen.....	79

8 TABELLENVERZEICHNIS

Tabelle 1 Verwaltungsschalen Funktionen [3]	29
Tabelle 2 Verwaltungsschale Eigenschaften [3]	29
Tabelle 3 Anforderungen Datenmodell	32
Tabelle 4 QoS Eigenschaften	43
Tabelle 5 Quality of Presentation Eigenschaften	48