Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Tobias Vielweber

Informationsmodellierung im OPC UA Adressraum für eine standardisierte Kommunikation am Beispiel eines Stückgutprozesses

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Tobias Vielweber

Informationsmodellierung im OPC UA Adressraum für eine standardisierte Kommunikation am Beispiel eines Stückgutprozesses

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Holger Gräßner
Zweitgutachter : Prof. Dr.-Ing. Florian Wenck

Abgegeben am 23. Mai 2018

Tobias Vielweber

Thema der Bachelorarbeit

Informationsmodellierung im OPC UA Adressraum für eine standardisierte Kommunikation am Beispiel eines Stückgutprozesses

Stichworte

Standardisierung, Industrie 4.0, Kommunikation, OPC UA, Informationsmodell, Companion Specification, Plug-and-Work

Kurzzusammenfassung

Diese Arbeit umfasst die Umsetzung einer standardisierten Kommunikationsschnittstelle über das Protokoll OPC UA. Für ein Labormodell wurde ein Informationsmodell nach einem Kommunikationskonzept für Industrie 4.0-Komponenten entwickelt. Diese wurde in einem OPC UA Server sowie in einem OPC UA Client eingebunden. Abschließend wurde die Umsetzung bewertet.

Tobias Vielweber

Title of the paper

Information modeling in the OPC UA address space for standardized communication using the example of a general cargo process

Keywords

Standardization, Industry 4.0, Communication, OPC UA, Information model, Companion Specification, Plug-and-Work

Abstract

This work covers the implementation of a standardized communication interface via the OPC UA protocol. For a laboratory model, information model based on a communication concept for Industry 4.0 components has been developed. This was integrated in an OPC UA server and in an OPC UA client. Finally, the implementation has been evaluated.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

BMBF Bundesministerium für Bildung und Forschung

BMWi Bundesministerium für Wirtschaft und Energie

BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien

DCOM Distributed Component Object Model

ERP Enterprise-Resource-Planning

IOSB-INA Institut für Optronik, Systemtechnik und Bildauswertung - Institutsteil für industrielle Automation

HMI Human-Machine Interface

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IEC International Electrotechnical Commission

MES Manufacturing Execution System

OLE Object Linking and Embedding

OPC Open Platform Communications

OPC UA OPC Unified Architecture

RAMI 4.0 Referenzarchitekturmodell Industrie 4.0

SCADA Supervisory Control and Data Acquisition

SDK Software Development Kit

SOAP Simple Object Access Protocol

SPS Speicherprogrammierbare Steuerung

TSN Time Sensitive Networking

TCP/IP Transmission Control Protocol/Internet Protocol

TLS Transport Layer Security

URI Unique Resource Identifier

VDMA Verband Deutscher Maschinen- und Anlagenbau

WS Web Service

XML Extensible Markup Language

ZVEI Zentralverband Elektrotechnik- und Elektronikindustrie

1 Einleitung

1.1 Stand der Technik

Die Innovationszyklen von digitalen Technologien steigen die letzten Jahrzehnte enorm an. Moderne Kommunikationsprotokolle basieren heute auf Ethernet. Durch sinkende Herstellungskosten von elektronischen Bauteilen besitzen Komponenten heute größtenteils eine Ethernet-Schnittstelle. Auch in Automatisierungslösungen sind Protokolle auf Ethernet-Basis mittlerweile der Standard. Unterschiedliche Hersteller oder Organisationen spezifizieren verschiedene Protokolle für den Datenaustausch von Automatisierungsprodukten wie z.B. PROFINET IO, EtherCAT oder Modbus. Ethernet spezifiziert Übertragungsraten mit bis zu 400 Gbit/sek[1]. Weit verbreitet in Automatisierungsprodukten sind heutzutage die Übertragungsraten von 100MBit/sek sowie von 1Gbit/sek. Im Vergleich zu Feldbussen wie z.B. PROFIBUS DP, mit einer maximalen Übertragungsrate von 12Mbit/sek, hat sich diese signifikant gesteigert.

Dies ermöglicht den Zugriff auf mehr Daten und schafft damit neue Möglichkeiten durch Kommunikation in der Automatisierung. Einzelne Teilaufgaben einer komplexen Aufgaben können nun auf mehrere Komponenten in einem Kommunikationsnetz verteilt werden. Dadurch werden Teilaufgaben einfacher und es können insgesamt komplexere Vorgänge realisiert werden. Ebenfalls ermöglicht dies eine flexible Anlagenstruktur und dadurch auch eine effizientere Nutzung der Ressourcen. Durch die Verteilung von Aufgaben wird die Kommunikation elementar wichtig für das Zusammenspiel mehrerer Komponenten. Werden Komponenten unterschiedlicher Hersteller in einer Automatisierungslösung verwendet, so ist es notwendig eine gemeinsame und herstellerübergreifende Kommunikationsbasis zu finden. Dabei empfiehlt es sich auf aktuelle Standards zu setzen. Eine Möglichkeit der herstellerübergreifenden Kommunikation bietet das Protokoll OPC UA.

Die Kommunikation über OPC UA ist nach dem Client-Server-Prinzip aufgebaut. Hierbei stellt ein Server Informationen und Dienste in einem Netzwerk mehreren Clients bereit. Ein Client kann diese von mehreren Server nutzen. Die Abbildung der Informationen und Dienste am OPC UA Server stellt somit die Schnittstelle einer Komponente dar. Diese kann jedoch von unterschiedlichen Herstellern, die einen OPC UA Server auf ihren Komponenten zur

[1] Spezifiziert in der IEEE 802.3bs am 06.12.2017, Quelle: http://www.ieee802.org/3/bs/, Zugriff: 02.05.2018

Verfügung stellen, variieren, sodass die Informationen zwar über OPC UA zugänglich, jedoch durch unterschiedliche Modelle beschrieben sind. Wird eine Komponente durch eine Komponente eines anderen Herstellers ausgetauscht, so muss das übergeordnete System auf das neue Modell angepasst werden. Es können eigene Informationsmodelle als Erweiterung der OPC UA Spezifikation erstellt werden, sodass die Informationen und Dienste einer Komponente über eine einheitliche Schnittstelle abgebildet werden. Wenn ein standardisiertes Informationsmodell durch Komponenten unterschiedlicher Hersteller bereitgestellt wird, so sind die Informationen und Dienste über eine einheitliche Art und Weise zugänglich. Dadurch kann die Schnittstelle einer Komponente und damit auch Anwendungen unabhängig von Geräten entwickelt werden. (vgl. VDMA und IOSB-INA, 2017, S. 14) Ebenfalls wird dadurch die Integration von Komponenten vereinfacht.

Wird ein standardisiertes Informationsmodell in Kooperation mit der OPC-Foundation definiert, so wird dieses Companion Specification genannt. Der VDMA (Verband Deutscher Maschinen- und Anlagenbau) treibt die Entwicklung von Companion Specifications in den unterschiedlichen Mitgliedsbranchen voran. Es wurden bereits einige Standards veröffentlicht. Endkunden fordern zunehmend von den Unternehmen des Maschinen- und Anlagenbaus die Verwendung der definierten Companion Specification. In vielen Maschinen und Anlagen wird das Automatisierungssystem SIMATIC S7-1500 der Firma Siemens AG verwendet. Die Steuerungsprogramme werden mit Hilfe der Software STEP 7 entwickelt. Mit der aktuellen Version besteht die Möglichkeiten ein eigenes oder durch eine Companion Specifikation definiertes Informationsmodell als Schnittstelle am OPC UA Server einer SIMATIC S7-1500 einzubinden.

1.2 Ziel der Arbeit

Das Ziel der Bachelorarbeit ist es, eine mögliche Umsetzung einer standardisierten Kommunikationsschnittstelle durch eine OPC UA Informationsmodellierung aufzuzeigen. Die Integration von Komponenten dadurch zu optimieren ist erstrebenswert und somit auch Ziel dieser Arbeit. Diese Bachelorarbeit dient als Leitfaden für die Definition und Umsetzung einer Informationsmodellierung. Durch die Verwendung von standardisierten Kommunikationsschnittstellen wird die Vision Plug-and-Work von Industrie 4.0 in der Kommunikation möglich. (vgl. VDMA und IOSB-INA, 2017, S. 5) Ein Informationsmodell kann als branchenweiter Standard für z.B. einen Typ einer Maschine definiert werden oder lediglich als Schnittstellenbeschreibung dienen. Das Fraunhofer IOSB-INA (Institut für Optronik, Systemtechnik und Bildauswertung - Institutsteil für industrielle Automation) und der VDMA definieren folgende Ziele für den Einsatz einer standardisierten Kommunikationsschnittstelle durch ein Informationsmodell:

> „Bei Inbetriebnahme und Umbau passen Systemintegratoren und Automatisierer heute manuell Steuerungsprogramme an, wozu sie üblicherweise auf Handbücher, Datenblätter und informell festgehaltene Informationen, die sich von Hersteller zu Hersteller unterscheiden, angewiesen sind. Zukünftig verwenden Maschinen- und Anlagenbauer die Companion Specification, um einen herstellerübergreifenden Informationsaustausch zu ermöglichen. Eine neue Maschine kann so einfacher in eine Anlage integriert werden, da standardisierte Informationen bei unterschiedlichen Herstellern gleichermaßen vorhanden sind. So können sich Komponenten, Maschinen und Anlagen einfach in Systeme des Endanwenders, wie z.B. MES, integrieren.“ (VDMA und IOSB-INA, 2017, S. 19)

Es wird eine mögliche Umsetzung einer Informationsmodellierung am Beispiel eines Labormodells durchgeführt. Hierbei werden die einzelnen Schritte der Umsetzung beschrieben. Zunächst werden grundlegende Lösungsansätze von Kommunikation in Industrie 4.0 aufgezeigt. Anschließend werden die Grundlagen des Protokolls OPC UA betrachtet. Auf Basis dieser Grundlagen wird dann ein Informationsmodell für ein Labormodell erstellt. Dieses Informationsmodell wird anschließend in die Steuerung des Labormodells eingebunden und durch den internen OPC UA Server bereitgestellt. Auf Basis des erstellten Informationsmodells wird dann ein OPC UA Client entwickelt. Dieser wird eine mögliche Umsetzung von Plug-and-Work aufzeigen. Zum Schluss werden die erlangten Erkenntnisse zusammengefasst und bewertet.

2 Grundlagen

Dieses Kapitel dient der Bildung von wichtigen Grundlagen. Zunächst wird die vierte industrielle Revolution und Kommunikationskonzepte von Industrie 4.0 betrachtet. Anschließend werden die Grundlagen von OPC UA untersucht.

2.1 Industrie 4.0

2.1.1 Industrielle Revolution

In der Geschichte gab es immer wieder industrielle Revolutionen durch neue Technologien. Ebenfalls zeigt die Geschichte, dass es Jahre dauern kann, bis sich die neu entdeckten Technologien in der Produktion etablieren. Die erste industrielle Revolution fand ca. zwischen 1760 und 1840 statt. Die neue Technologie war die Erfindung der Dampfmaschine. Durch eine nun angetriebene mechanische Produktion entstand somit die „Industrie 1.0“.

Die zweite industrielle Revolution fand im späten 19. Jahrhundert und im frühen 20. Jahrhundert statt. Durch die Nutzung der Elektrizität sowie die Erfindung des Fließbandes, wurde die Massenproduktion möglich. Darüber hinaus gab es nun neue Möglichkeiten in der Kommunikation. Telefon, Telegramme sowie eine wachsende Infrastruktur öffneten neue Wege in der Industrie. Diese neuen Möglichkeiten in der Produktion war die Basis für die zweite industrielle Revolution, „Industrie 2.0“.

Die dritte industrielle Revolution begann in den 1960er Jahren. Durch die Entwicklung von Halbleitern und die daraus entwickelten Computer konnte die Produktion durch z.B. eine Speicherprogrammierbare Steuerung (SPS) weiter automatisiert werden. Diese neue digitale Automatisierung der Produktion ermöglichte die dritte industrielle Revolution, „Industrie 3.0“. (vgl. Schwab, 2016, S.16-17)

Die vierte industrielle Revolution basiert auf den digitalen Technologien der dritten industriellen Revolution. Computer-Hardware, -Software und -Netzwerke sind keine neuen Technologien, jedoch werden sie im Vergleich zur dritten industriellen Revolution immer komplexer und integrierter. (vgl. Schwab, 2016, S.17-18) Durch die Benutzung dieser digitalen Technologien soll eine ganzheitliche Betrachtung der Wertschöpfungskette von der Idee, über die

Entwicklung und Produktion, bis hin zum Recycling verwirklicht werden. Grundlage hierfür ist die Bereitstellung aller relevanten Informationen durch Vernetzung aller an der Wertschöpfung beteiligten Instanzen. (vgl. Zehl, 2018) Ziel der ganzheitlichen Betrachtung der Wertschöpfungskette ist es, eine größtmögliche Steigerung der Flexibilität sowie der Produktivität zu erreichen.

Der Begriff „Industrie 4.0“ wurde durch die Forschungsunion der deutschen Bundesregierung geprägt. Im Nachhinein wurden die vorhergehenden industriellen Revolutionen mit einer Versionsnummer benannt, sodass die industriellen Revolutionen unter „Industrie 1.0-3.0“ zusammengefasst wurden. Das grundlegende Ziel der Bundesregierung ist es, mittelfristig die internationalen Wettbewerbspositionen deutscher Unternehmen gegen stark anwachsende Konkurrenz aus Asien, und zunehmend auch aus Südamerika, zu stärken. „Unternehmen, etwa aus China, steigern ihre Produktivität und Innovationskraft, zugleich beschleunigen sich die Innovationskreisläufe in vielen Technologiefeldern“. (Bundesministerium für Wirtschaft und Energie (BMWi), 2013, S.6) Das übergeordnete Ziel, die Wettbewerbspositionen deutscher Unternehmen international zu stärken, soll durch verschiedene Konzepte und neue Technologien erreicht werden. Die Interpretation und das Verständnis für die Herangehensweise und die Definition von Industrie 4.0 variieren zwischen den Branchen und sogar zwischen einzelnen Unternehmen. Damit das übergeordnete Ziel von Industrie 4.0 erreicht werden kann, müssen Politik und Wirtschaft zusammenarbeiten, um neue Konzepte, Technologien und Strategien entwickeln zu können.

Nachdem der Arbeitskreis Industrie 4.0 der Forschungsunion Wirtschaft - Wissenschaft des BMBF (Bundesministerium für Bildung und Forschung) im Oktober 2012 den Bericht „Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0“ veröffentlichte, schlossen sich im April 2013 die Verbände BITKOM (Bundesverband Informationswirtschaft, Telekommunikation und neue Medien), VDMA (Verband Deutscher Maschinen- und Anlagenbau) und ZVEI (Zentralverband Elektrotechnik- und Elektronikindustrie) mit dem BMWi (Bundesministerium für Wirtschaft und Energie) und dem BMBF zusammen und gründeten die „Plattform Industrie 4.0“ mit dem Ziel, über Verbandsgrenzen hinweg die Zusammenarbeit auf das übergeordnete Ziel voranzutreiben. (vgl. BMWi, 2018) Seit Gründung der Plattform Industrie 4.0 ist die Zahl der mitwirkenden Teilnehmer deutlich gestiegen, sodass nun in der Organisation und in den Arbeitsgruppen eine große Mehrzahl der Interessengruppen vertreten sind. Eine Arbeitsgruppe der Plattform Industrie 4.0 veröffentlichte 2015 eine erste Umsetzungsstrategie, das Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0).

2.1.2 Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0)

(vgl. Plattform Industrie 4.0, 2016, S. 3-20)

Das Referenzarchitekturmodell Industrie 4.0 soll Unternehmen dabei unterstützen, das Thema Industrie 4.0 strukturierter angehen zu können. Alle Elemente sind in einem dreidimensionalem Schichten- und Lebenszyklusmodell zusammengefasst. „Komplexe Zusammenhänge können so in kleinere, überschaubare Pakete aufgegliedert werden.“ (ZVEI, 2015, S. 1) Teilbereiche des Modells bestehen aus bereits existierenden Standards. Im Folgenden ist das dreidimensionale Modell abgebildet.

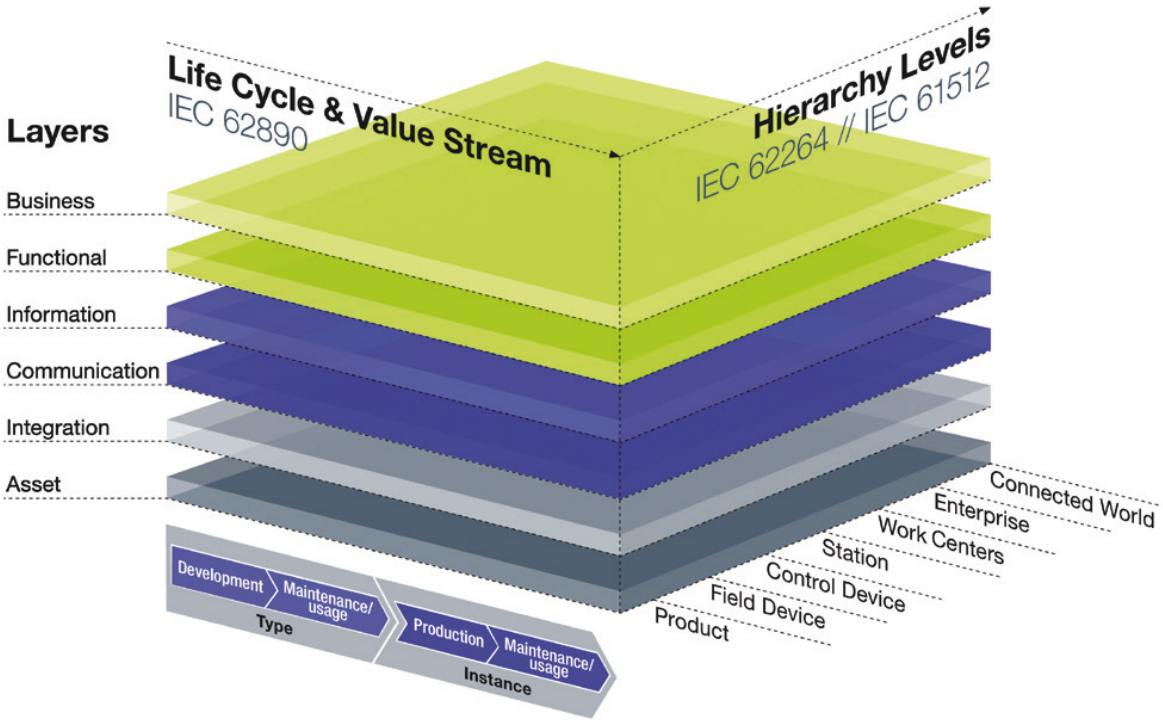

Abbildung 2.1: Referenzarchitekturmodell Industrie 4.0 (Plattform Industrie 4.0, 2015, S. 43)

Achse 1: Die Hierarchie

Die Strukturierung einer Automatisierungshierarchie ist in Industrie 3.0 durch die „Automatisierungspyramide“ abgebildet. Die Normungsorganisation IEC (International Electrotechnical Commission) definiert in der Normenreihe IEC 62264 die Schichten der Automatisierungspyramide. Die Automatisierungspyramide ist im Folgenden abgebildet.

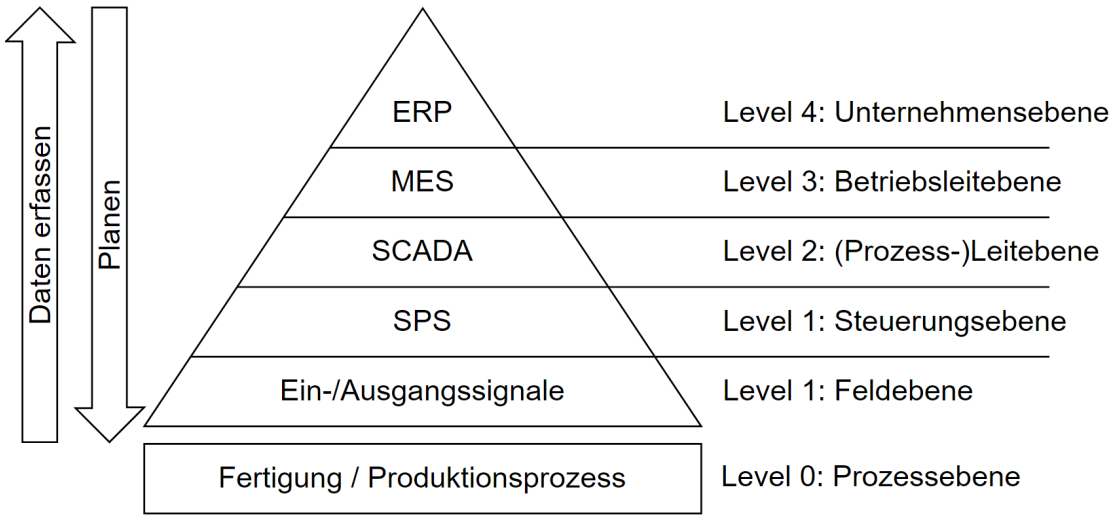

Abbildung 2.2: Automatisierungspyramide Industrie 3.0 (Wikipedia, 2018)

Die einzelnen Schichten beschreiben die Hierarchie der Funktionen einer Automatisierung innerhalb einer Fabrik oder Anlage. Das Produkt wird in diesem Modell nicht berücksichtigt. Eine Kommunikation findet nur zwischen den Hierarchieebenen statt. Die Hierarchie im RAMI 4.0 ist weitaus flexibler. Das Produkt ist nun Teil der Hierarchie. Kommunikation findet zwischen allen Hierarchieebenen statt. Dies ermöglicht eine flexible Maschinen- und Anlagennutzung. Aufgaben und Funktionen werden dezentral im Netz verteilt. Immer mehr rechenintensive Aufgaben werden heutzutage bereits von Servern über das Internet, sogenannte *Cloud Services*, bewältigt. Dieser Punkt ist nun unter *Connected World* ebenfalls Teil der Hierarchie.

Achse 2: Die Architektur

Während Achse 1 des RAMI 4.0 die Hierarchie bzw. die Anordnung einzelner Komponenten beschreibt, gliedert Achse 2 die Architektur einer Komponente und beschreibt die verschiedenen Aufgaben auf 6 verschiedenen Schichten. Die *Business*-Schicht beschreibt Geschäftsprozesse und Organisation. Alle Funktionen, welche eine Komponente spezifiziert, werden auf der *Functional*-Schicht beschrieben. Auf der *Information*-Schicht werden alle notwendigen Daten, welche die Komponente repräsentieren, gesammelt. Über die *Communication*-Schicht erfolgt der Zugriff auf die in der *Information*-Schicht gesammelten Daten. Die *Integration*-Schicht beschreibt den Übergang von der physischen in die digitale Welt. Auf der letzten Ebene, der *Asset*-Schicht, wird die physische Komponente beschrieben. Mithilfe der 6 Schichten kann die Architektur einer Komponente abgebildet werden.

Achse 3: Der Produktlebenszyklus

Eine wichtige Grundlage für die Umsetzung von Industrie 4.0 ist die Bereitstellung von relevanten Informationen aller an der Wertschöpfung beteiligten Instanzen zu jedem Zeitpunkt des Produktstatus. Aus diesem Grund wurde die dritte Achse in das RAMI 4.0 hinzugefügt. Diese Achse beschreibt 4 Phasen des gesamten Produktlebenszykluses. Es wird zwischen Phasen einer realen Instanz einer Komponente und Phasen der Typisierung einer Komponente unterschieden.

Die Kategorie *Type* ist unterteilt in die Phasen *Development* und *Maintenance/Usage*. Der erste Abschnitt *Development* beschreibt die Phase der Entwicklung und Konstruktion einer Komponente. Die Wartung eines Types einer Komponente durch z.B. Softwareupdates oder das Pflegen von Bedienungsanleitungen ist durch die Phase *Maintenance/Usage* beschrieben.

Die Kategorie *Instance* im Produktlebenszyklus ist unterteilt in die Phasen *Production* und *Maintenance/Usage*. Der erste Abschnitt der Kategorie *Production* beschreibt den realen Bau einer Komponente wie z.B. die Produktion, Seriennummer oder Produktionsdaten. Während der Phase *Maintenance/Usage* einer realen Komponente werden Modelle für z.B. Service, Wartung oder Recycling beschrieben. Alle 4 Phasen der Achse beschreiben somit den gesamten Produktlebenszyklus.

Die Industrie 4.0-Komponente

Auf Basis des Referenzarchitekturmodells Industrie 4.0 entwickelte eine Arbeitsgruppe der Plattform Industrie 4.0 ein erstes konkretes Modell für die Umsetzung einer Industrie 4.0-Komponente. Eine Industrie 4.0-Komponente ist in diesem Modell unterteilt in einen Gegenstand und in eine Verwaltungsschale, welches den Gegenstand digital mit hinreichenden Informationen beschreibt. Der Zugriff auf eine Industrie 4.0-Komponente erfolgt über die Verwaltungsschale. Bereits existierende Standards werden für die Beschreibung einer Komponente in die Verwaltungsschale eingeordnet. Die Verwaltungsschale setzt sich aus mehreren Teilmodellen zusammen. Ein Teilmodell der Verwaltungsschale kann z.B. bestehende Standards hinsichtlich Safety oder Security beschreiben. (vgl. BMWi, 2017) Die Produkteigenschaften, welche durch eine Verwaltungsschale beschrieben werden, sind in folgende Kategorien aufgeteilt.

- Identifikation der Komponente in Entwicklung, Logistik, Produktion, Vertrieb, etc.
- Kommunikation über Netzwerke für die Ansteuerung von Diensten oder das Auslesen von Daten
- Semantik: Standardisierte Datenablage

- Virtuelle Beschreibung über den gesamten Lebenszyklus
- Dienste und Zustände einer Komponente
- Standardfunktionen, welche herstellerunabhängig auf verschiedenen Produkten lauffähig sind
- Security

(vgl. ZVEI, 2016, S. 10)

Die Definition des Referenzarchitekturmodells Industrie 4.0 und des darauf aufbauenden Modells für eine Industrie 4.0-Komponente ist lediglich der erste Schritt für eine konkrete Umsetzungsstrategie. „Industrie 4.0 ist gegenwärtig noch nicht vollumfänglich beschrieben." (ZVEI, 2016, S. 11) Ein wichtiger Bestandteil der Umsetzung dieses Modells ist die Kommunikation von relevanten Daten und die Bereitstellung von Diensten über die Kommunikation.

Kommunikation

Alle Komponenten der einzelnen Hierarchieebenen der Achse 1 des RAMI 4.0 (Abb. 2.1) kommunizieren über die Hierarchieebenen hinweg miteinander. Damit dies in einem derart großen Rahmen möglich ist, stellen sich einige Anforderungen an die Kommunikation. Zunächst bedarf es einer einheitliche Sprache für den Austausch von Informationen. Die Installation und Inbetriebnahme der Kommunikation zwischen zwei Komponenten muss einfach sein. Wenn eine Kommunikation eingerichtet ist, können Dienste ausgeführt und Daten austauscht werden. (vgl. Plattform Industrie 4.0, 2016)

Plattform Industrie 4.0 hat ein Interaktionsmodell für eine Industrie 4.0-Komponente veröffentlicht. Dieses beschreibt, was die Komponenten in welcher Reihenfolge miteinander austauschen. Das Prinzip des Interaktionsmodell ist im folgenden Abbild dargestellt. (vgl. BMWi, 2016)

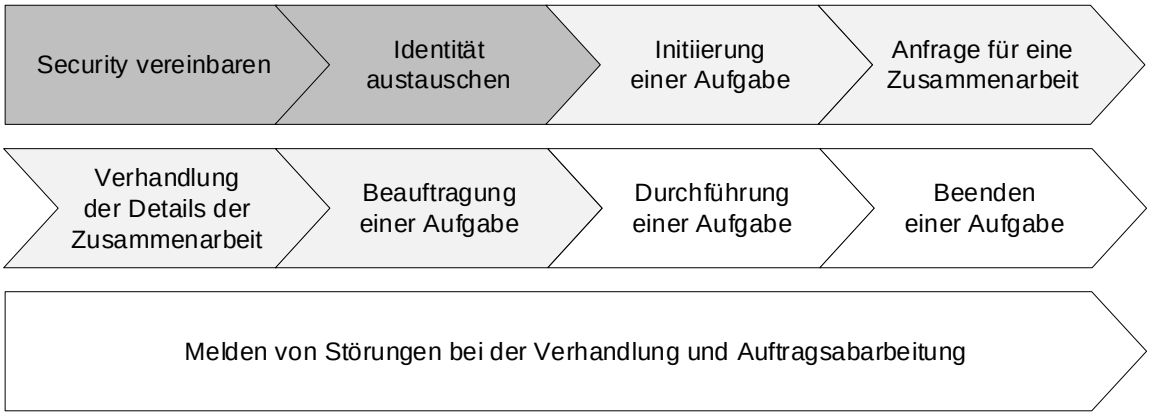

Abbildung 2.3: Interaktionsmodell Industrie 4.0-Komponenten

Durch eine solche Interaktion zwischen Komponenten kann eine flexible und effektive Nutzung von Ressourcen erreicht werden. Ein Kommunikationsmittel muss also unter anderem eine einfache Inbetriebnahme, Sicherheitsmechanismen und den Aufruf von Diensten unterstützen. Eine Möglichkeit, diese Kommunikation nach dem Interaktionsmodell für Industrie 4.0-Komponenten aufzubauen, bietet der nach der Normenreihe IEC 62541 genormte Kommunikationsprotokoll OPC UA.

2.2 OPC UA

2.2.1 OPC Entstehungsgeschichte

(vgl. Lange, 2014, S. 1-30)

Wie in allen Bereichen der Wirtschaft, wird auch in der Automatisierung stetig daran gearbeitet, effektiver, zeitsparender und kostengünstiger zu werden. Neben der Steigerung der Effektivität in der Softwareentwicklung durch Modularisierung und Kapselung von Softwarekomponenten, stellt sich ebenfalls die Frage nach einer standardisierten Schnittstelle auf Kommunikationsebene zwischen zwei Komponenten. Ohne eine solche standardisierte Schnittstelle ist die Integration einzelner Komponenten mit einem hohen Zeitaufwand, Programmieraufwand und damit auch mit einem hohen Kostenaufwand verbunden.

Durch die Einführung erster HMI (Human-Machine Interface) und SCADA (Supervisory Control and Data Acquisition) Programme für PC's zwischen 1989 und 1991 bekam der PC eine immer größere Bedeutung in der Prozessautomatisierung. Mit zunehmender Ausbreitung

dieser Produkte stieg die Zahl der dafür benötigten Kommunikationsprotokolle und Bussysteme enorm an.

Da immer mehr Ressourcen für die Entwicklung und die Wartung der Kommunikationstreiber benötigt wurden, gründeten im Jahr 1995 einige führende Firmen auf diesem Gebiet eine OPC Arbeitsgruppe, welche eine Lösung für die wachsende Komplexität der Kommunikationstreiber erarbeiten sollte.

Die Arbeitsgruppe setzte sich zum Ziel, auf Basis der DCOM (Distributed Component Object Model)-Technologie, welche eine Weiterentwicklung der OLE (Object Linking and Embedding)-Technologie von Microsoft ist, einen Standard für den Zugriff von Daten unter Windows-Rechnern zu erarbeiten. Hieraus entstand die Abkürzung OLE for Process Control (OPC). Die Abkürzung steht heute jedoch für Open Platform Communications (OPC). (vgl. OPC-Foundation, 2018) DCOM spezifiziert und beschreibt ein Objektmodell für das Implementieren verteilter Anwendungen nach dem Client-Server-Prinzip. So kann ein Client gleichzeitig die Funktionen mehrerer Server nutzen, und ein Server kann seine Funktionen mehreren Clients gleichzeitig zur Verfügung stellen.

Die Arbeitsgruppe stellte bereits im August 1996 die OPC Specification 1.0 vor. Im September 1996 wurde dann die OPC Foundation gegründet, die seither alle Spezifikations- und Marketingarbeiten koordiniert. Fast jährlich wurden neue Spezifikationen entwickelt oder bestehende überarbeitet.

Die einfache Einrichtung und Inbetriebnahme der Kommunikationsbeziehung lokaler OPC-Komponenten auf einem Windows-Rechner beschleunigten die Verbreitung dieser Technologie. Durch eine sehr komplexe DCOM Sicherheitseinstellung an einem Rechner wurde die Einrichtung der Kommunikationsbeziehung von OPC-Komponenten auf unterschiedlichen Windows-Rechnern deutlich erschwert.

Nach der Standardisierung von Extensible Markup Language (XML) im Jahr 1998 wurden neue Web-Service Technologien[1] wie z.B. das XML-Protokol SOAP (Simple Object Access Protocol) entwickelt. XML wurde für den plattformunabhängigen Datenaustausch spezifiziert. Aufgrund der erschwerten Einrichtung der Kommunikationsbeziehung von OPC-Komponenten auf unterschiedlichen Windows-Rechnern über das DCOM Modell, gründete die OPC Foundation im Jahr 2002 eine Arbeitsgruppe, welche den Zugriff auf OPC-Komponenten der Data Access Spezifikation unter Verwendung von XML und des XML-Protokol SOAP spezifizieren sollte. Dies ermöglichte den Zugriff auf OPC-Komponenten über das Internet von Betriebssystemen, welche keine DCOM-Unterstützung besaßen.

[1] „Ein Web-Service entspricht der XML-basierten Darstellung einer Anwendung oder Software-Komponente. Web-Services sind selbstbeschreibend; Schnittstelle und deren Metadaten sind getrennt. Konsument und Anbieter eines Web-Service sind lose gekoppelt; beide kommunizieren mittels XML-basierten Nachrichten über definierte Schnittstellen. Dadurch bleiben Details der Implementierung des Web-Service verborgen.“, Quelle: https://gi.de/informatiklexikon/web-services/, Zugriff: 22.03.2018

Nachdem klar wurde, dass auch alle weiteren Spezifikationen über Firewall-Grenzen hinweg auf Web-Services zugänglich gemacht werden müssen, wurde im Jahr 2003 die Arbeitsgruppe OPC Unified Architecture gegründet. Das Ziel war es, alle bereits entwickelten Spezifikationen plattformunabhängig und über eine einheitlich entwickelte Architektur bereitzustellen. Die bereits entwickelten Spezifikationen sind seitdem unter dem Begriff OPC-Classic zusammengefasst.

2.2.2 OPC UA Spezifikationen

(vgl. Lange, 2014, S. 105-116)

(vgl. Mahnke u. a., 2009, S. 11-17)

Die OPC UA Spezifikation besteht aus 13 Teilspezifikationen. Die Teilspezifikationen eins bis fünf wurden bereits 2006 vorgestellt. Bis 2009 wurden dann alle 13 Teilspezifikationen veröffentlicht. Die Aufteilung hat das Ziel, OPC UA Spezifikationen und Technologie, die sich ändern kann, zu trennen. Alle 13 Teilspezifikationen sind in drei Themenbereiche gegliedert: Core Specifications (Kernspezifikation), Access Type Specifications (Zugriffstypspezifikation) und Utility Specifications (Dienstspezifikation). In der folgenden Abbildung sind alle Teilspezifikationen dargestellt, gegliedert nach Themenbereichen.

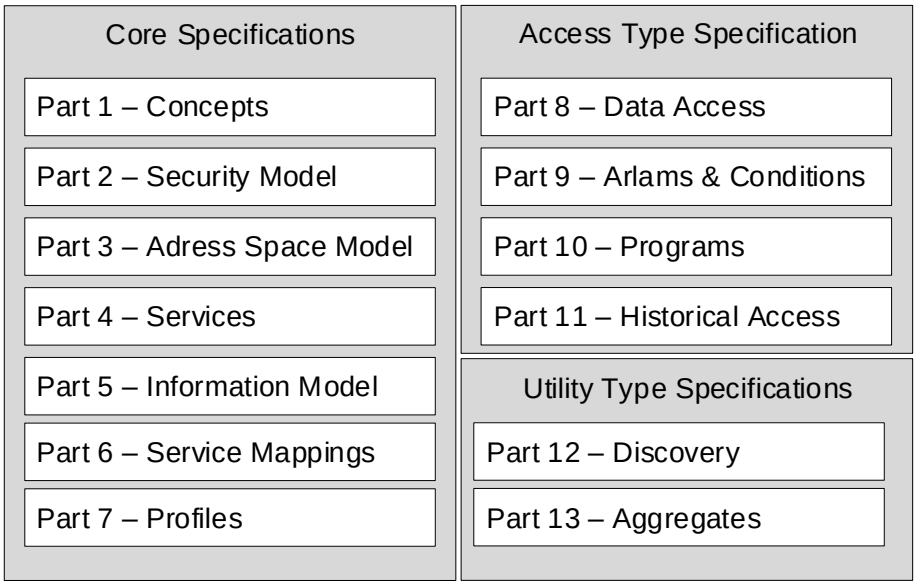

Abbildung 2.4: OPC UA Spezifikationen (nach Mahnke u. a., 2009, S. 12)

Im Folgenden wird ein kurzer Überblick der 13 Teilspezifikationen gegeben. Die detaillierte Betrachtung der Teilspezifikationen wird lediglich für die relevanten Themen, die für das Grundverständnis einer Informationsmodellierung nötig sind, vorgenommen.

Core Specifications

Die Kernspezifikationen beschreiben das Konzept und die Struktur des *AddressSpace* (Addressbereich) sowie der *Services* (Dienste).

Part 1: Concepts

Dieser Part beschreibt das Gesamtkonzept und die Zielsetzung von OPC UA. Hierbei werden die weiteren Parts der Kernfähigkeiten erläutert und in Zusammenhang gebracht.

Part 2: Security Model

Beim OPC UA Sicherheitsmodell wird darauf eingegangen, wie OPC UA bereits bestehende Sicherheitsstandards verwendet werden. Die eingeführten Sicherheitsverfahren sind Teil einer *Endpoint Description* eines Servers. Ein Server kann deshalb auch mehrere *Endpoint Desciptions* haben und somit unterschiedliche Sicherheitsstandards implementieren. Ein Client kann über den Service *Discovery* diese *Endpoints* finden, einen *Endpoint* auswählen, um dann auf Basis des gewählten *Endpoints* eine Kommunikation zum Server aufzubauen.

Part 3: Address Space Model

Ziel ist es, die reale Prozessumgebung sowie das Echtzeit Prozessverhalten auf einheitliche Art und Weise zugänglich zu machen. Die abgebildeten Objekte werden an einem OPC UA Server im *AdressSpace* (Adressraum) veröffentlicht. Die Objektinformationen werden hierbei durch *Nodes* (Knoten) abgebildet, welche dann über *References* (Referenzen) miteinander verbunden sind. Dieser Part beschreibt das Basismodell für die Modellierung von Informationen. Ein *Node* kann unterschiedliche vordefinierte *NodeClasses* haben, wie z.B. *Variable* oder *Object*.

Part 4: Services

Dieser Part spezifiziert die *Services* (Dienste), welche durch einen OPC UA Server bereitgestellt werden, wie z.B. die Funktion des *Browsen* durch den Adressraum, das Aufrufen einer Methode oder das Anlegen einer *Subscription*.

Part 5: Information Model

Das OPC UA-Informationsmodell beschreibt die standardisierten Typ- und Instanzknoten im Server Adressraum eines leeren OPC UA Servers, wie z.B. den Eintrittspunkt für Clients in den Adressraum des Servers oder die Basisdatentypen. Der Inhalt dieser Spezifikation kann als Grundlage verwendet werden, um alle Informationen für sämtliche Clients zu ermitteln. Das Informationsmodell beschreibt den konkreten Inhalt des Adressraum-Modells (Part 3) wie z.B. das Encoding eines Standarddatentyps.

Part 6: Service Mappings

Die *Service Mappings* spezifizieren die Abbildung der Datenkodierung, Sicherheits- und Transportprotokolle der beschriebenen Parts 2 bis 5 auf reale Technologien. Diese Trennung von abstrakter Beschreibung der Parts 2 bis 5 und der umgesetzten Technologie ermöglicht es, dass die Parts auch auf neu entwickelte Technologien durch ein Mapping umgesetzt werden können. Die folgende Abbildung zeigt die in Part 6 implementieren Technologien für den gesamten Kommunikationsweg.

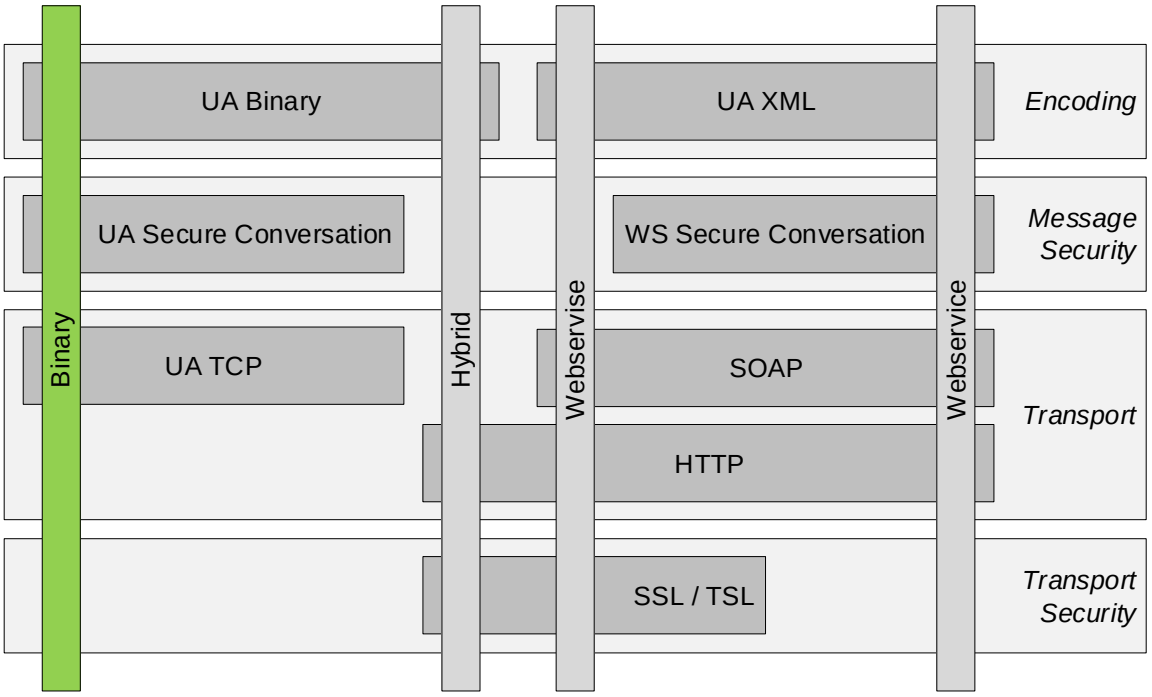

Abbildung 2.5: OPC UA Service Mapping (nach ascolab, 2018b)

Der Zugriff auf die OPC UA *Services* erfolgt über Webservice, Binärprotokoll oder aus einer Kombination von beidem. Es wird zwischen der *Message Security* und der *Transport Security* unterschieden. Beide Möglichkeiten spezifizieren die Sicherheit des Nachrichtenaustausches. Während *Message Security* die Sicherheit auf Nachrichtenebene vor dem Transport beschreibt, spezifiziert *Transport Security* den sicheren Datenaustausch auf der Transportebene. Auf der Applikationsebene gibt es zusätzliche Sicherheitsmechanismen, wie z.B. die Vergabe von Zugriffsrechten durch Benutzeranmeldung.

Als Transportprotokolle können TCP/IP (Transmission Control Protocol/Internet Protocol), HTTP (Hypertext Transfer Protocol) sowie HTTPS (Hypertext Transfer Protocol Secure) verwendet werden. Für die entsprechenden Zugriffe stellt der Part 6 (Service Mappings) zwei *Encodings* zur Verfügung. Die Daten können über *UA Binary* binär kodiert oder über *UA XML* als XML Stream kodiert werden. Der binäre Zugriff auf die OPC UA *Services* ist durch den geringen Overhead an Daten der performanteste Kommunikationsweg und muss von jedem Server bereitgestellt werden. Die unterschiedlichen Zugriffsmöglichkeiten ändern nichts an einer bereits bestehenden OPC UA Anwendung, da eine bestehende Anwendung flexibel durch den Part 6 auf die unterschiedlichen Technologien abgebildet werden kann.

Part 7: Profiles

OPC UA wurde so konzipiert, dass Anwendungen auf unterschiedlichen Endgeräten implementiert werden können. Sie können sich jedoch in Größe, Performance und Funktionsumfang unterscheiden. Deshalb sind OPC UA Funktionen in Profile eingeteilt. Die Profile definieren nützliche Subsets von OPC UA Funktionen. Ein Profil muss immer vollständig implementiert sein. Die Liste der unterstützten und verwendeten Profile werden beim Verbindungsaufbau zwischen Client und Server ausgetauscht und ermöglichen es dem Client festzustellen, ob die benötigten Funktionen vom Kommunikationspartner unterstützt werden.

Access Type Specifications

Die Spezifikationen der Access Type Specifications (Zugriffstypspezifikation) beschreiben den Datenzugriff auf Fähigkeiten der Kernspezifikationen (Part 1-7). Diese werden auf bestimmte Modelle des Datenzugriffs angewandt, wie z.B. den Datenzugriff über Data Access oder Historical Access.

Part 8: Data Access

Data Access beschreibt den Zugriff auf die Prozessvariablen über OPC UA, wie z.B. den lesenden Zugriff über Read oder den schreibenden Zugriff über Write.

Part 9: Alarm and Conditions

Alarm and Conditions beschreibt den Zugriff über OPC UA auf Alarme bzw. Meldungen. Die Konzepte der Quittierungs-Bedingung werden ebenfalls beschrieben, wie z.B. *acknowledgeable condition* (quittierbare Bedingung).

Part 10: Programs

Programs beschreibt den Zugriff über OPC UA auf zustandsgesteuerte Funktionen an einem Server. Diese zustandsgesteuerten Funktionen werden für das Steuern von Prozessen verwendet wie z.B. von einem Werkzeugmaschinenprogramm. Ein *Program* kann von einem Client gesteuert werden (z. B. Starten und Stoppen). Ebenso können Zwischenergebnisse durch *Events* an den Client zurückgegeben werden.

Part 11: Historical Access

Dieser Part beschreibt den Zugriff über OPC UA auf historische Daten. Es kann auf historische Prozess- und Eventdaten zugegriffen werden.

Utility Specifications

Die Utility Specifications (Dienstspezifikation) spezifizieren den Vorgang für das Finden und Erkennen von OPC UA Servern.

Part 12: Discovery

Der Part *Discovery* spezifiziert die Zusammenarbeit von Client und Server. Es wird der Vorgang beschrieben, wie ein Client den Server findet und sich mit ihm verbinden kann. Hierfür stellt der Server einen *Discovery* Server zur Verfügung, der allgemeine Informationen zu diesem Server bereitstellt. So können beispielsweise die in Part 2 definierten *Endpoint Descriptions* ausgetauscht und kommuniziert werden.

Part 13: Aggregates

Dieser Part spezifiziert die Erweiterung der *Core Specifications* für Berechnungen (Aggregates) von z.B. Minimum, Maximum, Mittelwert. Die Verwendung wird beschrieben für Echtzeitdaten sowie für historische Daten.

2.2.3 Informationsmodellierung

(vgl. Lange, 2014, S. 116-165)

(vgl. Mahnke u. a., 2009, S. 19-81)

Die Basisdienste bzw. die Kernspezifikationen (Part 1-7) enthalten lediglich die Information, wie Daten dargestellt werden können. Es werden die bereitzustellenden Informationen auf verschiedenste Art und Weise von unterschiedlichen Unternehmen zur Verfügung gestellt. Beispielhaft kann dies an dem Energiemessgerät SENTRON PAC3200 der Siemens AG dargestellt werden. Alle Messgrößen sind u. a. in Registern abgelegt und über Modbus TCP lesbar bzw. schreibbar. Es ist genau beschrieben, welcher Wert in welchen Registern abgelegt ist. (vgl. Siemens AG, 2008, S. 39-43) Bei Energiemessgeräten anderer Hersteller sind

die Daten möglicherweise anders ablegt und sogar über ein anderes Protokoll zugänglich gemacht.

Die Integration wird durch unterschiedliche Abbildungen nicht vereinfacht. Ein möglicher Weg dies zu optimieren ist es, standardisierte Informationsmodelle für OPC UA zu definieren und somit Dienste und Informationen über eine einheitliche Art und Weise bereitzustellen. Heute werden bereits Informationsmodelle für den einheitlichen Datenaustausch über OPC UA verwendet. So auch bei dem RFID-System SIMATIC RF600 der Siemens AG. Die RFID-Lesegeräte dieser Serie stellen u. a. über den internen OPC UA Server die Dienste und Informationen über das Informationsmodell „OPC UA for AutoID Companion Specification“ zur Verfügung. (vgl. Siemens AG, 2018b, S. 283 ff.) Die Daten sind somit über ein standardisiertes Informationsmodell über OPC UA veröffentlicht. Weitere Hersteller von RFID-Lesegeräten welche dieses Informationsmodell verwenden, veröffentlichen die selben Dienste und Informationen, sodass der Zugriff auf das RFID-Lesegerät der gleiche ist.

Informationsmodelle erweitern die Kernspezifikationen um z.B. eigene Datentypen oder Objekttypen. Dadurch kann eine hierarchische Objektstruktur von Typen oder Instanzen im Vorfeld beschrieben werden. Der Zugriff auf die erweiterten Kernspezifikationen erfolgt weiterhin über die Zugriffsspezifikationen. Für eine Informationsmodellierung werden Standardelemente aus der OPC UA Spezifikation benutzt. Alle Elemente einer Informationsmodellierung wie z.B. Datentypen, Objekttypen oder Variablentypen werden in einer XML-Datei gespeichert. Die wichtigsten Elemente werden im Folgenden genauer untersucht.

Nodes

Der in Part 3 definierte Adressraum stellt eine Sammlung von Informationen dar. Eine Information wird immer als *Node* (Knoten) abgebildet. Ein *Node* kann z.B. eine Variable oder ein Objekt darstellen. Die Summe aller Informationen bzw. aller *Nodes* bilden den Adressraum. Die *Nodes* werden über *Refenrences* miteinander verbunden. Der ursprüngliche *Node* ist hierbei der *Source Node* und der referenzierte *Node* der *Target Node*. Untergeordnete *Nodes* werden ebenfalls Kinder und die übergeordneten *Nodes* Eltern genannt. Es gibt unterschiedliche Typen der Referenzierung, die durch *ReferenceTypes* dargestellt werden.

Ein *Node* ist immer eine Instanz einer *NodeClass*. In der OPC UA Spezifikation sind die folgenden *NodeClasses* definiert:

NodeClass	Beschreibung
Variable	Diese Klasse stellt eine Variable dar
Method	Diese Klasse stellt eine Funktion dar
Object	Diese Klasse stellt ein Objekt dar
View	Diese Klasse stellt eine Ansicht einer Teilmenge von *Nodes* dar
DataType	Diese Klasse stellt den Datentyp eines *Value* Attribut einer Variable dar
VariableType	Diese Klasse stellt den Typ einer Variable dar
ObjectType	Diese Klasse stellt den Typ eines Objektes dar
ReferenceType	Diese Klasse stellt den Typ einer Referenz dar

Tabelle 2.1: NodeClass

Die Auswahl der *NodeClasses* ist fest definiert und darf nicht verändert werden. Es werden die folgenden Grafiken für die beschriebenen *NodeClasses* in Übersichtsdiagrammen verwendet.

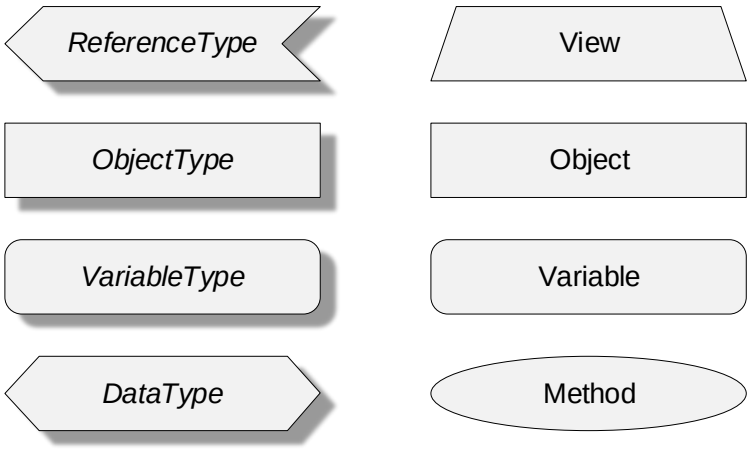

Abbildung 2.6: Grafische Abbildung der *NodeClasses* (nach Mahnke u. a., 2009, S. 328)

Jeder *Node* besteht aus Attributen und Referenzen. Einige Attribute können auch optional sein. Es müssen folgend Attribute von jeder *NodeClass* veröffentlicht werden:

Attribute	Beschreibung
NodeId	Eindeutige Kennung eines *Nodes* im Adressraum
NodeClass	Gibt die Klasse der *Node* Instanz an
BrowseName	Der Name des *Node* in Klartext
DisplayName	Der Anzeigename des *Node* für den Benutzer
Description	Optional: Beschreibung oder Bedeutung des *Node*

Tabelle 2.2: Node Attributes

Die *NodeId* ist die eindeutige Adresse eines *Nodes* im Adressraum. Der Zugriff eines Clients auf *Nodes* eines Servers erfolgt entweder über *Random Access* oder über *Browsing*. Beim *Random Access* erfolgt der Zugriff über die im Vorfeld bekannte und am Server eindeutige *NodeId*. Ist die Struktur eines Servers und damit auch die eindeutige Kennung eines *Nodes* im Vorfeld nicht bekannt, so wird ein *BrowsePath* der *Node*, also der Pfad über die *BrowseNames* der *Nodes*, in eine eindeutige *NodeId* umgewandelt, wodurch dann wieder der Zugriff über *Random Access* erfolgen kann. Eine *NodeId* wird als folgende Struktur abgebildet.

Attribute	Beschreibung
NamespaceIndex	Index des *Namespaces*, in dem sich der *Node* befindet
IdentifierType	Gibt den Typ der Identifikation eines *Nodes* an, wie z.B. numerisch oder als Zeichenkette
Identifier	Beschreibt den im Namespace eindeutigen Identifikator des *Node*

Tabelle 2.3: NodeId Attributes

Ein *Namespace* (Namensraum) steht für eine URI (Unique Resource Identifier). Die *NamespaceURI* ist eine Zeichenkette und beschreibt die für den im *Namespace* enthaltenen *Nodes* verantwortliche Ressource. In den OPC UA Services wird eine *Namespace URI* durch eine Zahl dargestellt, um die Effizienz in der Übertragung und der Verarbeitung zu steigern. Die Zuordnung der *NamespaceIndexes* zu den *NamespaceURIs* ist in einer Tabelle am Server abgelegt.

Jeder Server muss den *Namespace* mit dem Index *0* und der *NamespaceURI http://opcfoundation.org/UA/* bereitstellen, in welchem die Kernspezifikationen, wie z.B. die Basisdatentypen, der OPC UA Spezifikationen definiert sind. Die grundlegende Organisation der OPC UA Spezifikation im *AdressSpace* ist im folgenden Abbild dargestellt.

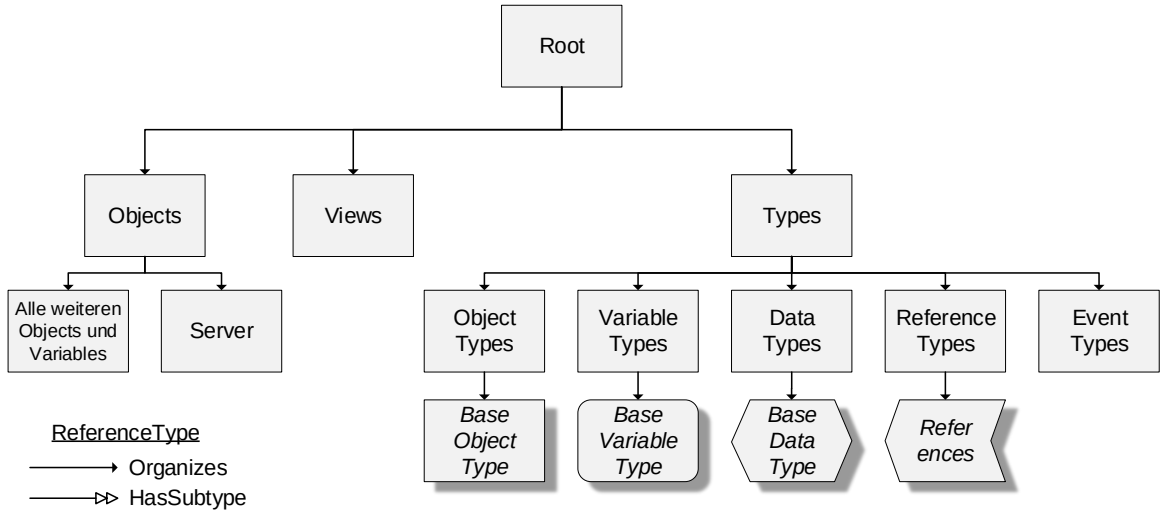

Abbildung 2.7: Standard *Nodes* im OPC UA *AdressSpace* (nach Mahnke u. a., 2009, S. 114)

Für einen erstellten *Node* in einer Informationsmodellierung kann eine Modellierungsregel definiert werden, sodass z.B. ein Kind eines Objekttyps beim Instanziieren optional oder verpflichtend ist. Dies ermöglicht eine Flexibilität beim Erstellen einer allgemeingültigen Schnittstelle über eine standardisierte Informationsmodellierung, da dann einige Komponenten nur optional am jeweiligen OPC UA Server vorhanden sein müssen und somit die Informationsmodellierung den gesamten Ausbau einer Komponente beschreiben kann. Im *AddressSpace* eines OPC UA Servers werden alle *Nodes* als *Target Node* eines Standard *Nodes* (Abb. 2.7) veröffentlicht. Diese werden über die entsprechenden Referenzen miteinander verbunden.

Referenzen

Eine *Reference* stellt eine Beziehung zwischen zwei *Nodes* dar. Es gibt unterschiedliche Typen einer *Reference*, welche als *ReferenceTypes* beschrieben sind. *ReferenceTypes* werden nicht instanziiert, sodass keine *NodeClass* (Tabelle 2.1) definiert ist, die eine *Reference* darstellt. Jede *NodeClass* hat eine Sammlung von *References*. Die OPC UA Spezifikation definiert die zugelassenen Sammlungen der *References* für die entsprechenden *NodeClasses* mit dem dazugehörigen *Types*. Für das Erstellen einer Informationsmodellierung ist die richtige Wahl einer Referenz wichtig, da für den jeweiligen *Node* der korrekte *ReferenceType* verwendet werden muss. Folgende Referenztypen sind in Part 5 der OPC UA Spezifikation definiert.

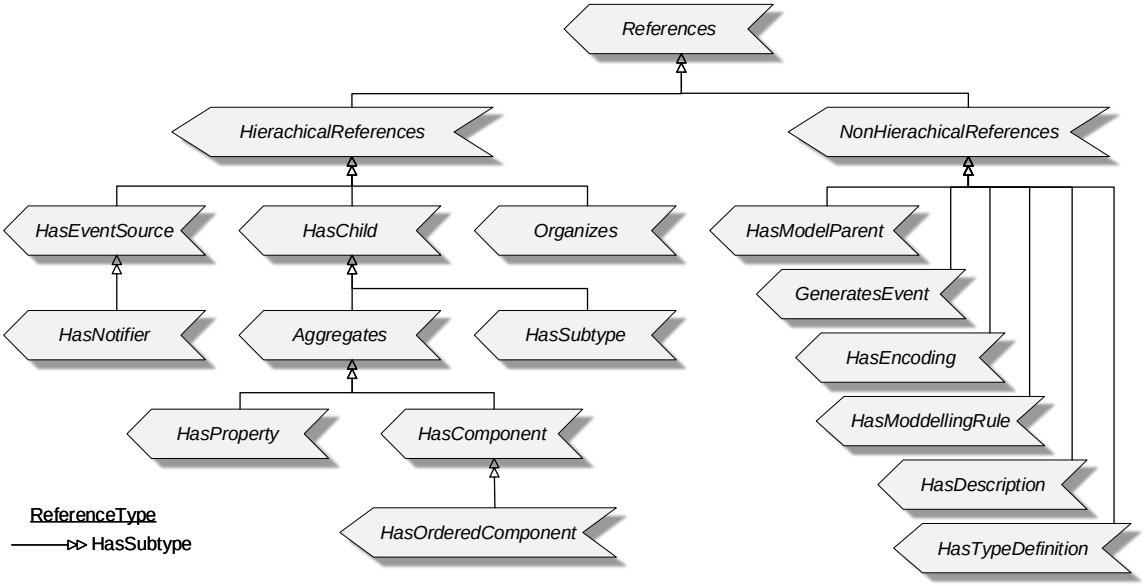

Abbildung 2.8: ReferenceTypes Hierachie (nach Mahnke u. a., 2009, S. 335)

Es wird grundlegend zwischen zwei Typen von Referenzen unterschieden, *HierachicalReferences* und *NonHierachicalReferences*. Beide erben von der abstrakten Klasse *References*. *HierachicalReferences* stehen für die Beziehungsart *Association*. Diese *Associations* (Zuordnung oder Assoziation) dienen dem Aufbau der Informationsarchitektur wie z.B. die der *Node* Hierachie im *AdressSpace*. *NonHierachicalReferences* stehen für die Beziehungsart *Dependency* (Abhängigkeit). Durch eine *Dependency* wird angegeben, dass der *Source Node* den *Target Node* verwendet oder von ihm abhängig ist. Eine Ausnahme dieser Kategorisierung bildet die Referenz *HasSubtype*, die für die Veröffentlichung der Vererbungshierarchie verwendet wird, und somit dem Aufbau der Informationsarchitektur dient und eine Abhängigkeit darstellt. Im Folgenden ist der Verwendungszweck von häufig verwendeten Referenztypen aufgeführt.

ReferenceType	Verwendungszweck
HasTypeDefinition	Verweist auf einen *VariableType* oder *ObjectType* und gibt somit die Klasse der Instanz an.
Organizes	Organisation von *Nodes* im *AdressSpace*. *Source Node* muss eine *View* oder ein *Object* vom Typ *FolderType* sein.
HasSubtype	Veröffentlichung einer Vererbungshierarchie von *DataTypes*, *VariableTypes*, *ObjectTypes* und *ReferenceTypes*. Der *Target Node* erbt vom *Source Node*.
HasProperty	Der *Target Node* beschreibt Eigenschaften des *Source Node*. Es werden meist nicht veränderliche Eigenschaften von *Objects*, *Variables* und *Methods* dargestellt.
HasComponent	Der *Target Node* ist Teil des *Source Node*. Es werden Beziehungen von *Objects*, *Variables* und *Methods* dargestellt.

Tabelle 2.4: ReferenceType Verwendungszweck

Variablen

Variablen repräsentieren im OPC UA *AdressSpace* Werte. Es wird zwischen *Properties* (Eigenschaften) und *DataVariables* (Daten Variable) unterschieden. *DataVariables* repräsentiert den Wert an sich. *Properties* stellen zusätzliche Eigenschaften anderer *Objects* und *Variables* sowie anderer *Nodes* dar. Durch zusätzliche Eigenschaften einer Variable kann so z.B. die Einheit und der Bereich einer Temperatur-Variable veröffentlicht werden.

Nodes der *NodeClass Variable* (Tabelle 2.1) müssen zusätzlich zu den Attributen einer *NodeClass* (Tabelle 2.2) folgende Attribute veröffentlichen:

- *Value*
- *DataType*
- *AccessLevel*
- *UserAccessLevel*

Das Attribut *Value* stellt den Wert der Variable dar. Der Datentyp des Wertes der Variable wird über das Attribut *DataType* veröffentlicht. Das Attribut *DataType* ist vom Datentyp *NodeId* (Tabelle 2.3) und stellt somit die *DataTypeId* dar. Jede Variable hat eine *Reference* vom Typ *HasTypeDefinition* zu einem *Target Node VariableType*. Ein *VariableType* definiert die Bedeutung einer Variable wie z.B. der Variablentyp *AnalogItemType*. Dadurch können z.B. die Eigenschaften wie Einheit oder Bereich des Wertes einer Variable veröffentlicht werden.

Datentypen

Datentypen beschreiben den Typ des Value Attribut einer Variable. Jeder Datentyp hat mindestens eine Referenz vom Typ *HasEncoding*. Somit beschreibt ein Datentyp das *Encoding* eines *Value Attributs*. Die in Part 6 (Service Mapping) der OPC UA Spezifikation beschriebenen möglichen Daten-Encodings (Abbildung 2.5) sind das *UA Binary Encoding* sowie das *UA XML Encoding*.

Die OPC UA Spezifikation definiert einige Basisdatentypen im *NamespaceIndex 0*. Diese werden nicht im *AdressSpace* veröffentlicht, da jeder Client die *Encodings* dieser Datentypen kennen muss. Basisdatentypen sind z.B.:

- Double
- Float
- Int32
- String
- ExtensionObject
- NodeId

Eine Struktur beschreibt einen komplexen Datentyp. Es können eigene Strukturen als Subtyp des abstrakten Datentyp *Structures*, welcher vom Datentyp *BaseDataType* (Abb. 2.7) erbt, beschrieben werden. Da ein Client eventuell anwenderspezifische Strukturen nicht kennt, werden die Variablen des Datentyps dieser Struktur einheitlich in einem *ExtensionObject* veröffentlicht. Die Struktur *ExtensionObject* kann von jedem Client gelesen werden und veröffentlicht ebenfalls die *DataTypeId* der anwenderspezifischen Struktur.

Alle Strukturen, die nicht durch Strukturen der Basisdatentypen beschrieben sind, werden am Server in einem *TypeDictionary* veröffentlicht. Mit der Beschreibung der Struktur durch das *TypeDictionary* und die durch das *ExtensionObject* veröffentlichte *DataTypeId*, kann die Struktur aus dem *ExtensionObject* von einem Client decodiert werden. Kennt ein Client im Vorfeld die Beschreibung einer anwenderspezifischen Struktur, so kann diese ohne das Lesen des *TypeDictionary* decodiert werden. Bei dieser Methode muss ein Client die gesamte Struktur lesen und decodieren, um auf einzelne Elemente der Struktur zuzugreifen.

Es gibt jedoch noch weitere Möglichkeiten, eine komplexe Datenstruktur einem Client bereitzustellen. Die Struktur kann ebenfalls in einem *ObjectType* oder einem *VariableType* abgebildet werden. Diese referenzieren mithilfe des *ReferenceTypes HasComponent* (Tabelle 2.4) andere einfache Variablen mit einfachen Datentypen, wodurch dann die anwenderspezifische Struktur durch den hierarchischen Aufbau durch Referenzen abgebildet wird. Instanziiert man eine solche komplexe Variable eines *VariableTypes*, der eine Struktur durch

Referenzierung der einzelnen Elemente dieser Struktur abbildet, so stellt das *Value* Attribut dieser Variable nicht die Struktur dar. Ein großer Vorteil dieser Methode ist, dass ein Client das *Encoding* einer anwenderspezifischen Struktur nicht kennen muss, da die Abbildung der Struktur aus Basisdatentypen besteht. Auch kann auf einzelne Elemente der Struktur zugegriffen werden, ohne die gesamte Struktur lesen zu müssen. Werden alle Elemente der Struktur gelesen, ist die Konsistenz der Daten nicht sichergestellt, da sich die Werte anderer Elemente der Struktur während des Lesezugriffs eines *Nodes* ändern können.

Weiter gibt es die Möglichkeit, die beiden bereits beschriebenen Abbildungen einer Struktur zu kombinieren. Hierbei wird eine komplexe Variable, welche die Struktur über die hierarchischen Referenzierungen abbildet, instanziiert. In das Attribut *DataType* dieser komplexen Variable wird dann die *DataTypeId* der erstellten Struktur geschrieben, die ein Subtyp des abstrakten Datentyps *Structures* ist. Die anwenderspezifische Struktur wird nun über die hierarchische Referenzierungen des *VariableType* sowie über das *ExtensionObject* des *Value* Attributes der Variable dargestellt. Die einfachen Variablen der hierarchischen Referenzierung müssen folglich nicht zwangsweise Elemente der encodierten Struktur des *ExtensionObject* darstellen. Für die Integrität der Daten dieser Abbildung einer Struktur ist somit der bereitstellende Server verantwortlich.

Objekte

Durch Objekte können Systemkomponenten, Softwareobjekte oder reale Objekte abgebildet werden. Es werden durch Referenzierung von Teilkomponenten und Eigenschaften durch den *ReferenceType HasComponent* und *HasProperty* (Tabelle 2.4) Objekte dargestellt. Objekte können somit auch der Organisation von Variablen oder anderen Objekten, ähnlich wie eine Ordnerstruktur auf einem Rechner, dienen. Jedes Objekt hat eine Referenz vom Typ *HasTypeDefinition* zu einem *Target Node ObjectType* und ist somit eine Instanz eines Objekttyps. Objekte werden durch die Referenz *Organizes* von dem *Node Objects* (Abb. 2.7) referenziert.

3 Entwickeln und Einbinden einer Informationsmodellierung

In diesem Kapitel wird eine mögliche Umsetzung einer Kommunikation über standardisierte Kommunikationsschnittstellen zwischen Industrie 4.0-Komponenten mittels OPC UA untersucht. Zunächst wird der Aufbau und die einzelnen Funktionen des Labormodells beschrieben. Auf Basis der Strukturierung und der Teilbeschreibungen wird dann eine Schnittstellenbeschreibung als OPC UA Informationsmodellierung für die einzelnen Komponenten erstellt. Die Einbindung und Implementierung dieser Schnittstelle wird dann für einen Client und einen Server umgesetzt.

3.1 Anlagenbeschreibung

3.1.1 Aufbau Stückgutprozess

Die Entwicklung und Objektbeschreibung wird an einem Labormodell durchgeführt. Das Labormodell ist ein Stückgutprozess und besteht aus drei Teilprozessen. Ein Stückgut ist ein Transportgut, welches einzeln und am Stück transportiert wird. Jeder Teilprozess symbolisiert eine ganze Maschine. Der Stückgutprozess besteht aus einem Warenlager, einem Transportsystem und einer Produktionsmaschine. Da das im Labor verwendete Stückgut einem Reifen ähnlich sieht, wird die Produktionsmaschine als Reifenproduktionsmaschine bezeichnet.

Die Anlage besteht aus elektronischen und pneumatischen Bauteilen. Der Aufbau ist so entwickelt, dass das Transportsystem das Warenlager und die Reifenproduktionsmaschine erreicht. Ein Stückgut wird manuell in das Warenlager hinzugefügt. Fertig produzierte Reifen werden manuell von einem Ablageplatz der Reifenproduktionsmaschine „abtransportiert“. Der Aufbau des Stückgutprozesses ist in folgendem Abbild dargestellt.

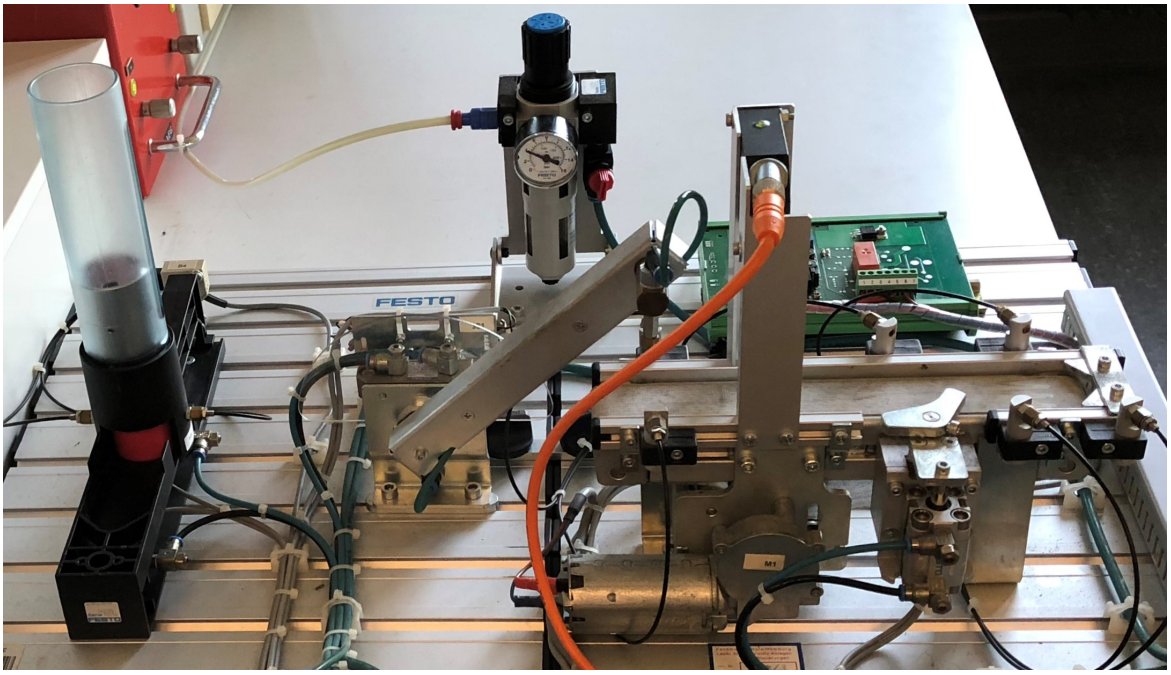

Abbildung 3.1: Stückgutprozess Anlagenübersicht

3.1.2 Teilprozesse

Warenlager

Das Warenlager (links in Abb. 3.1) besteht aus einem Übergabeplatz, einem Ausschieber und einem zylinderförmigen Ablageplatz. Der Ausschieber schiebt über einen pneumatischen Zylinder das unterste des im Ablageplatz liegenden Stückguts bis hin zum Übergabeplatz. Von dem Übergabeplatz kann das Transportsystem dann das abgelegte Stückgut abholen. Ein neues Stückgut muss zum Einlagern manuell in den zylinderförmigen Ablageplatz eingeführt werden. Der Ablageplatz kann zehn Stückgüter einlagern.

Reifenproduktionsmaschine

Die Reifenproduktionsmaschine (rechts in Abb. 3.1) besteht grundlegend aus einem Laufband, einem Drehschieber und einigen Sensoren. Auf diesem Laufband gibt es vier mögliche Positionen für das Stückgut. Position eins ist der Übergabeplatz für ein Stückgut, das durch das Transportsystem abgeliefert wird. An Position zwei findet die eigentliche „Reifenproduktion“ statt. Da dieses Modell keine komplexe Produktion oder einen Reifenproduktionsprozess ermöglicht, wird die Reifenproduktion durch eine dreisekündige Pause des Stückguts

an Position zwei symbolisiert. Ist ein Stückgut als Ausschuss gekennzeichnet, dann wird die Produktionspause nicht vorgenommen.

Nach abgeschlossenem Prozess an Position zwei wird das Stückgut an Position drei transportiert. Der Drehschieber an Position drei öffnet den Weg für genau ein Stückgut zu Position vier, wenn diese frei ist. An Position vier liegt dann ein Stückgut bereit zur Abholung. Da an dieser Stelle der Laboraufbau zu Ende ist, muss das Transportgut manuell abgeführt werden.

Transportsystem

Das Transportsystem (mittig in Abb. 3.1) befördert ein Stückgut vom Übergabeplatz des Warenlagers zum Übergabeplatz der Reifenproduktionsmaschine. Ein pneumatischer Schwenkarm ermöglicht die Rotationsbewegung vom Warenlager zum Transportsystem. Liegt ein Stückgut am Übergabeplatz des Warenlagers bereit, so fährt der Schwenkarm zum Warenlager und saugt das Stückgut an einer kleinen Düse mittels Unterdruck an. Meldet ein Sensor, dass das Stückgut erfolgreich angesaugt wurde, dann fährt der Schwenkarm mit dem Stückgut zum Übergabeplatz der Reifenproduktionsmaschine. Gibt die Reifenproduktionsmaschine eine Freigabe für die Ablage des Stückguts, so legt der Schwenkarm das Stückgut am Übergabeplatz ab und fährt in seine initialisierte Position zurück.

3.1.3 Komponentenbeschreibung

Das in Kapitel 2.1.2 vorgestellte RAMI 4.0 beschreibt eine Umsetzungsstrategie für Industrie 4.0. Eine erste konkrete Umsetzung dieser Strategie beschreibt die Industrie 4.0-Komponente. Kommunikation und die einheitliche Bereitstellung von Diensten ist ein elementarer Bestandteil der Beschreibung einer Industrie 4.0-Komponente. Ebenfalls ist durch die Plattform Industrie 4.0 bereits ein Konzept für die Interaktion von Industrie 4.0-Komponenten entwickelt worden (Abb. 2.3). Es werden die Schichten *Functional*, *Information* und *Communication* der Architekturachse des RAMI 4.0 untersucht. Zunächst wird eine abstrakte Beschreibung der Funktionalitäten der Teilsysteme auf der *Functional* Schicht vorgenommen.

Die drei Teilsysteme Warenlager, Reifenproduktionsmaschine und Transportsystem benötigen alle eine Identifikation als Komponente. Alle allgemeinen Informationen einer Komponente können also einheitlich beschrieben werden. Nach dem Interaktionsmodell (Abb. 2.3) benötigt eine Komponente eine eigene Auftragsverwaltung. Diese hat die Funktion, Auftragsanfragen zu verhandeln, zu beantworten und je nach Rückmeldung zu bearbeiten. Im Rahmen dieser Arbeit wurde das Konzept auf das Labormodell angewandt. Die Übersicht der Auftragskette ist im folgenden Abbild dargestellt.

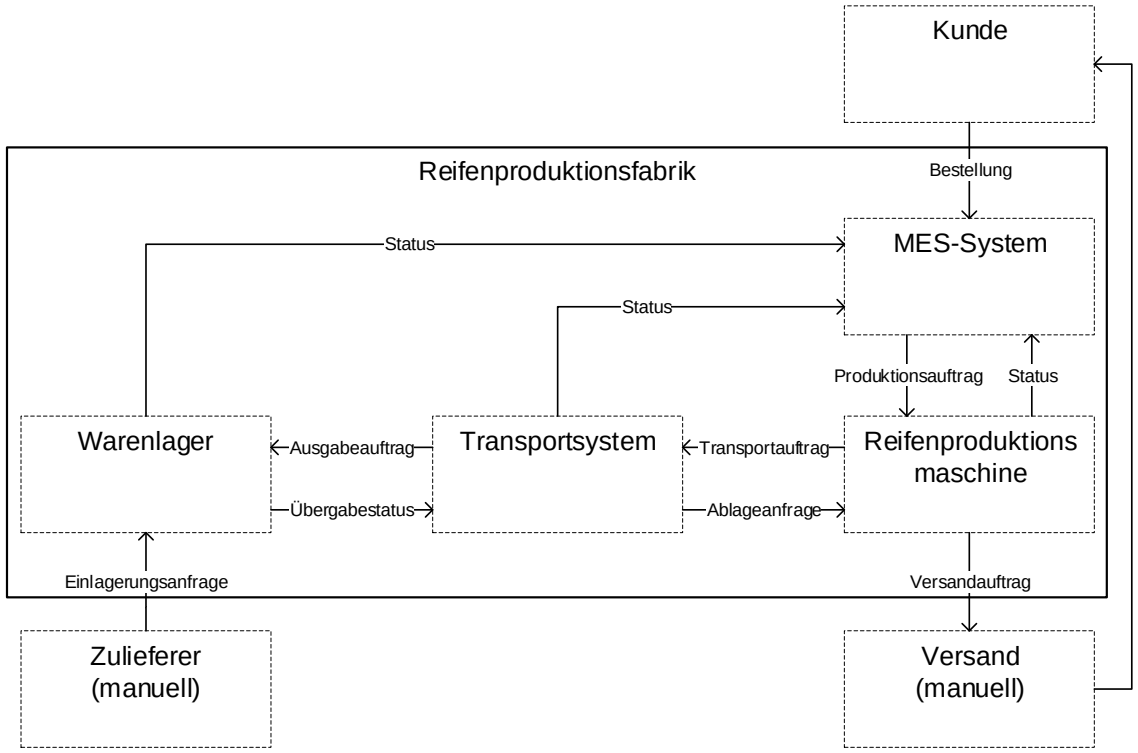

Abbildung 3.2: Fabrik-Auftragssystem Übersicht

Alle Auftragspfeile der Abbildung 3.2 stellen den im Interaktionsmodell (Abb. 2.3) beschriebenen Prozess (Initiieren einer Aufgabe, Anfrage für eine Zusammenarbeit, Verhandlung der Details und Beauftragung einer Aufgabe) dar. Der Pfeilursprung stellt den Initiator der Kommunikation bzw. den Client dar. Die Kommunikation findet jedoch in beide Richtungen statt. Um die Auftragsprozesse für das Beispiel des Stückgutprozesses zu vereinfachen, werden die Anfrage für eine Zusammenarbeit und Verhandlung der Details in der Funktion Beauftragung einer Aufgabe zusammengefasst. Wird ein Auftrag angenommen, so gibt die Funktion der Auftragsverwaltung ein TRUE zurück. Das Warenlager muss in der Auftragsverwaltung folgende Dienste und Informationen veröffentlichen:

- Stückgut einlagern
- Ausgabeauftrag erteilen
- Übergabestatus veröffentlichen
- Statusinformationen vom Warenlager

Da das Warenlager immer nur das unterste Stückgut ausschieben kann, fordert das Transportsystem solange neue Ausgabeaufträge an, bis alle Transportaufträge abgeschlossen

wurden. Wird ein Stückgut transportiert, wofür kein Transportauftrag vorliegt, so wird dieser als Ausschuss markiert. Das Transportsystem muss in der Auftragsverwaltung folgende Dienste und Informationen veröffentlichen:

- Transportauftrag erteilen
- Statusinformationen vom Transportsystem

Die Reifenproduktionsmaschine muss in der Auftragsverwaltung folgende Dienste und Informationen veröffentlichen:

- Produktionsauftrag erteilen
- Stückgut übergeben
- Statusinformationen der Reifenproduktionsmaschine

Die aufgelisteten Dienste und Informationen sind in der Architekturschicht *Functional* des RAMI 4.0 einzuordnen. Damit diese über die Kommunikation mittels OPC UA bereitgestellt werden können, muss die Architekturschicht *Information* beschrieben werden.

3.2 Informationsmodellierung

Durch eine Informationsmodellierung im OPC UA Adressraum werden standardisierte Kommunikationsschnittstellen beschrieben. Da OPC UA plattformunabhängig entwickelt wurde, können die unterschiedlichsten Hersteller von Automatisierungslösungen diese Informationsmodellierung an einem OPC UA Server bereitstellen. Wenn eine Informationsmodellierung in Kooperation mit der OPC Foundation entwickelt wurde, so wird diese als Companion Specification bezeichnet. Es entstehen immer mehr Companion Specifications in den unterschiedlichsten Branchen. Einige entwickelte Companion Specification sind z. B. EUROMAP für den Datenaustausch von Spritzgussmaschinen oder AIM für den Datenaustausch von AutoID Geräten wie RFID Lesegeräte.

Die beschriebenen Datentypen, Variablentypen, Objekttypen und die hierarchische Anordnung der Nodes werden in einer XML-Datei gespeichert. In Modellierungseditoren können diese XML-Dateien importiert, editiert und exportiert werden. So können z. B. Objekttypen, welche in der Informationsmodellierung beschrieben sind, beliebig oft instanziiert werden. Die Instanzen werden im *Objects* Ordner (Abb. 2.7) im OPC UA Adressraum aufgelistet. Die Verweise von OPC UA Variablen und Methoden auf Funktionen und Prozessvariablen einer Steuerung wird herstellerspezifisch realisiert. Nach fertiger Bearbeitung der Informationsmodellierung und der Verweise auf Steuerungs-Variablen und -Funktionen wird diese

Schnittstellenbeschreibung als XML-Datei exportiert. Diese kann dann in der Steuerung eingebunden und am OPC UA Server veröffentlicht werden. Der beschriebene Stückgutprozess wird mit einer SIMATIC S7-1500 Steuerung der Siemens AG automatisiert.

3.2.1 Modellierungseditor

Es wird der OPC UA Modellierungseditor SiOME der Siemens AG verwendet. Dieser kann kostenlos im *Siemens Industry Online Support* unter der Beitrags-ID 109755133[1] geladen werden. Der Modellierungseditor ist wie folgt aufgebaut.

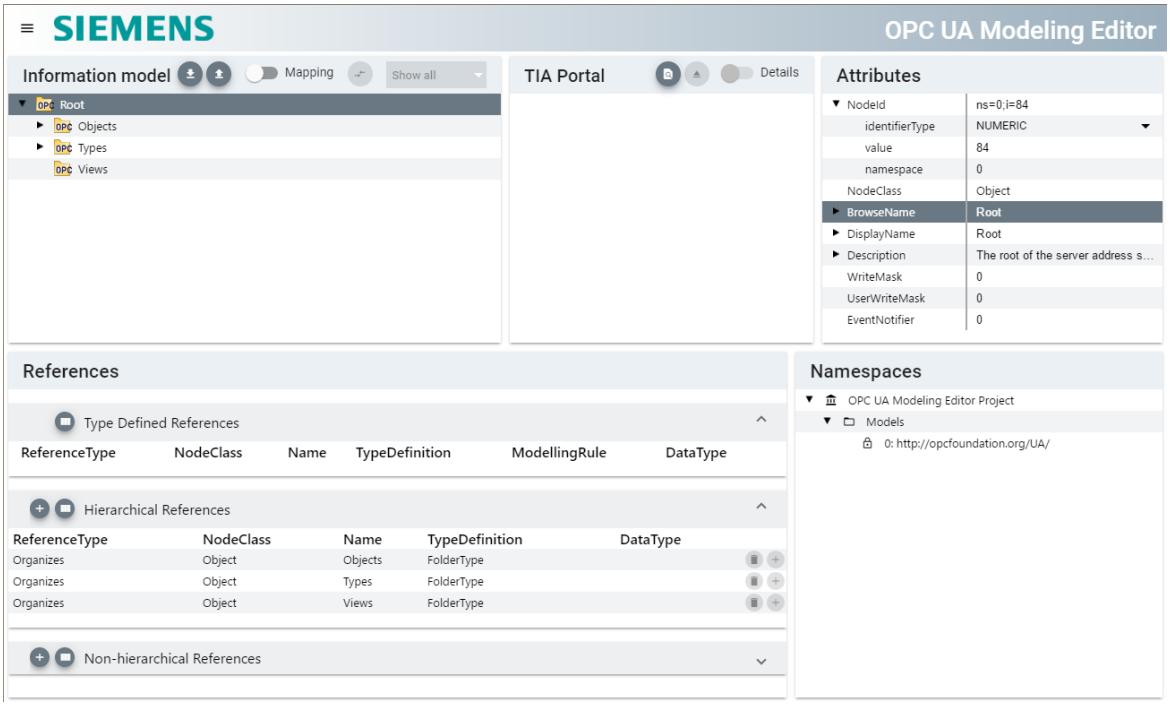

Abbildung 3.3: SiOME Oberfläche

In dem Fenster *Information model* wird der OPC UA Adressraum angezeigt. Über die Import-Funktion können bestehende Informationsmodellierungen als XML-Datei importiert werden. Der neue Namespace wird dann im Fenster *Namespaces* angezeigt. Über die Export-Funktion werden alle Nodes, die nicht im *Namspaceindex* 0 sind, in eine XML-Datei exportiert. Es können die Mappings aller Nodes angezeigt und editiert werden. Das Mapping auf eine Variable in einer SIMATIC S7-1500 Steuerung wird über eine Erweiterung in der XML-Datei für die instanziierte Variable realisiert. Im folgenden Listing ist die Abbildung einer

[1] Quelle: https://support.industry.siemens.com/cs/ww/de/view/109755133, Zugriff: 11.04.2018

instanziierten und gemappten Variable in XML dargestellt. Das Mapping auf die Steuerungsvariable wird in Zeile 11 beschrieben.

```
1  <UAVariable DataType="String" NodeId="ns=1;i=1083"
2    BrowseName="2:TestVariable">
3    <DisplayName>TestVariable</DisplayName>
4    <References>
5      <Reference ReferenceType="HasProperty" IsForward="false">ns=1;i=1080
6        </Reference>
7      <Reference ReferenceType="HasTypeDefinition">i=68</Reference>
8    </References>
9    <Extensions>
10     <Extension>
11       <si:VariableMapping>"Global_DB"."Variable"</si:VariableMapping>
12     </Extension>
13   </Extensions>
14 </UAVariable>
```

Stimmt der Datentyp der OPC UA Variable nicht mit dem Datentyp der Steuerungsvariablen überein, so wird die Variable nicht am OPC UA Server bereitgestellt. Das Mapping einer OPC UA Methode auf eine Funktion in der Steuerung wird über folgende Erweiterung realisiert.

```
1 <Extensions>
2   <Extension>
3     <si:MethodMapping>"Instance_DB".Method</si:MethodMapping>
4   </Extension>
5 </Extensions>
```

Beim Mapping von Methoden müssen die Ein- und Ausgangsparameter der Steuerungs-Funktion mit denen der OPC UA Methode übereinstimmen. Falls diese nicht übereinstimmen, kann die Methode am OPC UA Server der Steuerung nicht bereitgestellt werden. Beim Aufrufen der fehlerhaft gemappten Methode OPC UA Server wird eine Fehlermeldung am Client ausgegeben.

Im Fenster *TIA Portal* (Abb. 3.3) können Datenbausteine und Variablenlisten eines geöffneten Projektes angezeigt werden. Wenn die Funktion Mapping im *Information model* Fenster aktiviert ist, können Variablen oder Methoden per Drag-and-Drop auf Variablen und Methoden im OPC UA Adressraum gemappt werden. Die beschriebenen Erweiterungen für das Mapping werden beim Export in der XML-Datei hinzugefügt.

Das *Attributes* Fenster (Abb. 3.3) zeigt alle Attribute, die eine im *Information model* Fenster markierten Node veröffentlicht. Einige Attribute können in diesem Fenster editiert werden. Das *Namespace* Fenster zeigt die Tabelle der Zuordnung von NamespaceIndex und

Namespace-URI. Hier können neue Namespaces erstellt und editiert werden. Das Fenster *References* gibt eine Übersicht aller Referenzen eines im *Information model* Fenster markierten Node. Neue Datentypen, Variablentypen und Objekttypen werden direkt im *Information model* Fenster angelegt und dann in den entsprechenden Fenstern editiert. Mithilfe dieses Tools wird eine Informationsmodellierung für den beschriebenen Stückgutprozess entwickelt.

3.2.2 Allgemeine Maschinenbeschreibung

Auf Grundlage der bereits bestehenden Companion Specification EUROMAP 77[2] wurde die allgemeine Maschinenbeschreibung des Labormodells entwickelt. Da die Companion Specification für Spritzgussmaschinen entwickelt ist, enthält diese ebenfalls eine allgemeine Objektbeschreibung einer Maschine. Daher wurde diese als Grundlage der Informationsmodellierung des Labormodells verwendet. Es wurden die Objekttypen *MachineInformationType*, *MachineConfigurationType* und *MachineStatusType* für die allgemeine Beschreibung einer Komponente definiert. Der Objekttyp *MachineInformationType* veröffentlicht Informationen zu der Maschine. Die hierarchischen Referenzen des Objekttyps sind in folgender Tabelle aufgelistet.

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition
HasProperty	Variable	ControllerName	String	PropertyType
HasProperty	Variable	DeviceManuel	String	PropertyType
HasProperty	Variable	DeviceRevision	String	PropertyType
HasProperty	Variable	HardwareRevision	String	PropertyType
HasProperty	Variable	Manufacturer	String	PropertyType
HasProperty	Variable	SerialNumber	String	PropertyType
HasProperty	Variable	SoftwareRevision	String	PropertyType

Tabelle 3.1: MachineInformationType

Der Objekttyp *MachineConfigurationType* veröffentlicht Informationen zum Standort und zur Zeit (Tabelle 3.2).

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition
HasProperty	Variable	LocationName	String	PropertyType
HasComponent	Method	SetServerTime	-	-

Tabelle 3.2: MachineConfigurationType

[2]Quelle: http://www.euromap.org/en/euromap77/, Zugriff: 09.04.2018

Die Methode *SetServerTime* hat als Eingabeparameter die Variable *Time* vom Datentyp DateTime und als Ausgabeparameter den Rückgabewert *RetVal* vom Datentyp Bool. Dieser zeigt an, ob die Funktion am Server erfolgreich die Zeit des Systems gesetzt hat. Der Objekttyp *MachineStatusType* veröffentlicht Methoden und Informationen zum aktuellen Status der Maschine (Tabelle 3.3).

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition
HasComponent	Variable	MachineMode	String	BaseDataVariableType
HasComponent	Method	ActivateSleepMode	-	-
HasComponent	Method	DeactivateSleepMode	-	-

Tabelle 3.3: MachineStatusType

Die Methoden *ActivateSleepMode* und *DeactivateSleepMode* besitzen keine Eingabeparameter und einen Ausgabeparameter *RetVal* vom Datentyp Bool als Bestätigung, ob die Funktion den neuen Status des *SleepMode* akzeptiert hat.

3.2.3 Komponentenbeschreibung

In Kapitel 3.1.3 wurden bereits die Dienste und Informationen für die Auftragsverwaltung der drei Teilsysteme definiert. Für die Informationsmodellierung sind diese in Objekttypen abgebildet worden. Die Dienste der jeweiligen Auftragsverwaltungen wurden als Methode implementiert. Dies bietet mehrere Vorteile. Zum einen ist die Konsistenz der Übergabeparameter beim Methodenaufruf sichergestellt, auch wenn eine beliebige Struktur nicht als *ExtensionObject* übergeben wird. Zum anderen wird ein direktes Antworten auf Auftragsanfragen durch die Rückgabeparameter ermöglicht, womit kein aufwendiger Handshake implementiert werden muss. Die Definition der Objekttypen der jeweiligen Auftragsverwaltung ist im Folgenden beschrieben.

Reifenproduktionsmaschine

Die Auftragsverwaltung der Reifenproduktionsmaschine ist durch den Objekttyp *TireProductionJobType* abgebildet (Tabelle 3.4).

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition
HasComponent	Variable	JobCounterRed	Uint	BaseDataVariableType
HasComponent	Variable	JobCounterBlack	Uint	BaseDataVariableType
HasComponent	Variable	JobCounterSilver	Uint	BaseDataVariableType
HasComponent	Variable	DailyProductionRed	Uint	BaseDataVariableType
HasComponent	Variable	DailyProductionBlack	Uint	BaseDataVariableType
HasComponent	Variable	DailyProductionSilver	Uint	BaseDataVariableType
HasComponent	Variable	DailyDefectProduction	Uint	BaseDataVariableType
HasComponent	Method	AddTireProductionJob	-	-
HasComponent	Method	Delivery	-	-

Tabelle 3.4: TireProductionJobType

Die Methode *Delivery* hat als Eingabeparameter die Variable *ActiveTransportJob* vom Typ ActiveTransportJobType und als Ausgabeparameter die Variable *RetVal* vom Typ Bool. War die Ablagefrage eines aktiven Transportgutes an die Reifenproduktionsmaschine erfolgreich, so liefert die Variable *RetVal* ein TRUE zurück. Die Methode *AddTireProductionJob* hat folgende Ein- und Ausgabeparameter.

Name	Datentyp
Input	
JobID	String
NumberOfPieces	UInt16
TireType	Byte
Output	
RetVal	Bool

Tabelle 3.5: AddTireProductionJob Parameter

Der Objekttyp *TireProduction_IMM_MES* dient als Schnittstelle der Reifenproduktionsmaschine zu anderen Maschinen oder einem Manufacturing Execution System (MES) (Tabelle 3.6).

ReferenceType	NodeClass	BrowseName	TypeDefinition
HasComponent	Object	JobManagement	TireProductionJobType
HasComponent	Object	MachineConfiguration	MachineConfigurationType
HasComponent	Object	MachineInformation	MachineInformationType
HasComponent	Object	MachineStatus	MachineStatusType

Tabelle 3.6: TireProduction_IMM_MES

Transport

Die Auftragsverwaltung des Transportsystems wird durch den Objekttyp *TransportJobType* abgebildet. Die hierarchischen Referenzen des Objekttyps zeigt die folgende Tabelle.

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition
HasComponent	Variable	JobCounterRed	Uint	BaseDataVariableType
HasComponent	Variable	JobCounterBlack	Uint	BaseDataVariableType
HasComponent	Variable	JobCounterSilver	Uint	BaseDataVariableType
HasComponent	Variable	AcitveTransportJob	ActiveTransportJobType	ActiveTransportJobType
HasComponent	Method	AddTransportJob	-	-

Tabelle 3.7: TransportJobType

Für die Variable *AcitveTransportJob* wird eine Struktur als Datentyp ActiveTransportJobType und ein Variablentyp ActiveTransportJobType definiert. Somit wird die dritte Möglichkeit zur Darstellung von Strukturen (Kapitel 2.2.3 - Datentypen) angewandt. Die Elemente des Variablentyps bilden die Struktur durch Referenzierungen von einfachen Variablen ab. Es wird ebenfalls die gesamte Struktur in das *Value* Attribut der Variable geschrieben. Die Struktur beinhaltet eine Variable der Struktur TireType. Beide Strukturen sind wie folgt definiert.

Name	Datentyp
ActiveTransportJobType	
AimJobID	String
TireMaterial	TireType
TireType	
ID	String
TireType	Byte

Tabelle 3.8: ActiveTransportJobType und TireType

Mit der Methode *AddTransportJob* werden Transportaufträge zum Transportsystem hinzugefügt. War die Anfrage erfolgreich, so gibt die Funktion ein TRUE zurück. Die Funktion hat folgende Ein- und Ausgabeparameter.

Name	**Datentyp**
Input	
JobID	String
NumerOfPieces	UInt16
TireType	Byte
DestinationDeviceID	String
Output	
RetVal	Bool

Tabelle 3.9: AddTransportJob Parameter

Im Labor sind Stückgüter unterschiedlicher Farbe vorhanden. Die Variable *TireType* nimmt den Wert 1 für rot, 2 für schwarz, 3 für silber und 4 für Ausschuss an. Der Reifentyp Ausschuss wird in die Variable *TireType* eins Stückgutes eingetragen, wenn das Transportsystem einen Baustein transportieren muss, für den kein Transportauftrag vorliegt. Dies ist bedingt durch das Warenlager, welches lediglich das unterste Stückgut des Lagers ausschieben kann. Als Schnittstelle des Transportsystems zu anderen Maschinen oder einem MES dient Objekttyp Transport_IMM_MES (Tabelle 3.10).

ReferenceType	**NodeClass**	**BrowseName**	**TypeDefinition**
HasComponent	Object	JobManagement	TransportJobType
HasComponent	Object	MachineConfiguration	MachineConfigurationType
HasComponent	Object	MachineInformation	MachineInformationType
HasComponent	Object	MachineStatus	MachineStatusType

Tabelle 3.10: Transport_IMM_MES

Warenlager

Der Objekttyp WarehouseJobType veröffentlicht Methoden und Informationen zur aktuellen Auftragsverwaltung des Warenlagers (Tabelle 3.11).

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition
HasComponent	Variable	StoredTireMaterialRed	Uint	BaseDataVariableType
HasComponent	Variable	StoredTireMaterialBlack	Uint	BaseDataVariableType
HasComponent	Variable	StoredTireMaterialSilver	Uint	BaseDataVariableType
HasComponent	Variable	DeliveryPlaceLoaded	Boolean	BaseDataVariableType
HasComponent	Variable	DeliveryPlaceObject	TireType	TireType
HasComponent	Method	StoreTireMaterial	-	-
HasComponent	Method	ProvideTireMaterial	-	-

Tabelle 3.11: WarehouseJobType

Der Datentyp TireType der Variable DeliveryPlaceObject ist in Tabelle 3.8 definiert. Mit der Methode *StoreTireMaterial* wird ein neues Stückgut in das Warenlager eingelagert. War die Anfrage erfolgreich, so gibt die Funktion ein TRUE zurück. Die Funktion hat die folgenden Ein- und Ausgabeparameter.

Name	Datentyp
Input	
ID	String
TireType	Byte
Output	
RetVal	Bool

Tabelle 3.12: StoreTireMaterial Parameter

Mit dem Aufrufen der Methode *ProvideTireMaterial* wird eine Ausgabeanfrage gestellt. War die Anfrage erfolgreich, so gibt die Funktion ein TRUE zurück. Liegt der Baustein auf dem Übergabeplatz bereit, so wird die Variable *DeliveryPlaceLoaded* TRUE. Die benötigten Informationen zu dem Baustein werden in der Variable *DeliveryPlaceObject* hinterlegt. Der Objekttyp Warehouse_IMM_MES dient als Schnittstelle des Transportsystems zu anderen Maschinen oder einem MES (Tabelle 3.13).

ReferenceType	NodeClass	BrowseName	TypeDefinition
HasComponent	Object	JobManagement	WarehouseJobType
HasComponent	Object	MachineConfiguration	MachineConfigurationType
HasComponent	Object	MachineInformation	MachineInformationType
HasComponent	Object	MachineStatus	MachineStatusType

Tabelle 3.13: Warehouse_IMM_MES

Es wurden alle benötigten Dienste und Informationen der drei Teilprozesse für eine Informationsmodellierung beschrieben. Diese wird nun mithilfe des Tools SiOME in einer XML-Datei abgebildet.

3.2.4 Informationsmodellierung der Teilprozesse

Die beschriebenen Dienste und Informationen der drei Teilprozesse zur Maschinenbeschreibung und für die Auftragsverwaltung wurden in Objekttypen abgebildet. Es wird der Ablauf und die einzelnen Teilschritte der Umsetzung dieser Beschreibungen in eine OPC UA Informationsmodellierung erläutert.

Erstellen eines Namespaces

Zu Beginn muss ein neuer Namespace angelegt werden. Die URI des Namespace beschreibt die verantwortliche Ressource. Da kein reales Projekt hinter der Modellierung steht, werden beispielhafte URIs verwendet. Eine neuer Namespace wird in dem Fenster Namespace über Rechtsklick auf Models angelegt.

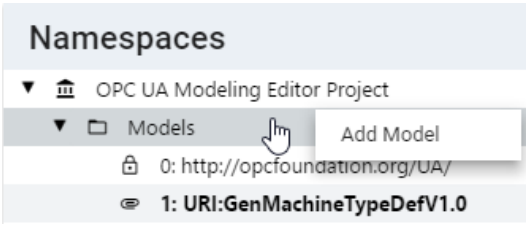

Abbildung 3.4: Erstellen eines Namespace

Markiert man einen Namespace, so werden alle Nodes, die sich in dem Namespace befinden, im *Information model* Fenster ebenfalls markiert. Es können nun Nodes im Fenster *Information model* zu dem Namespace hinzugefügt werden. Jeder Teilprozess wird durch

einen eigenen Namespace beschrieben, da auch die verantwortlichen Ressourcen sich für die Teilprozesse in der Realität unterscheiden würden. Die allgemeinen Maschineninformationen werden in dem Namepsace *URI:GenMachineTypeDefV1.0* beschrieben, das Warenlager wird im Namespace *URI:WarehouseV1.0* modelliert, die Reifenproduktionsmaschine wird im Namespace *URI:TireProductionV1.0* beschrieben, und das Transportsystem wird im Namespace *URI:TransportV1.0* modelliert.

Erstellen eines Objekttyps

Zum Erstellen einer hierarchischen Struktur können Objekttypen definiert werden. Neue Objekttypen werden als Subtyp mit der Referenz *HasSubtype* des Nodes *BaseObjectType* angelegt. Durch Rechtsklick auf *BaseObjectType* lassen sich neue Objekttypen hinzufügen.

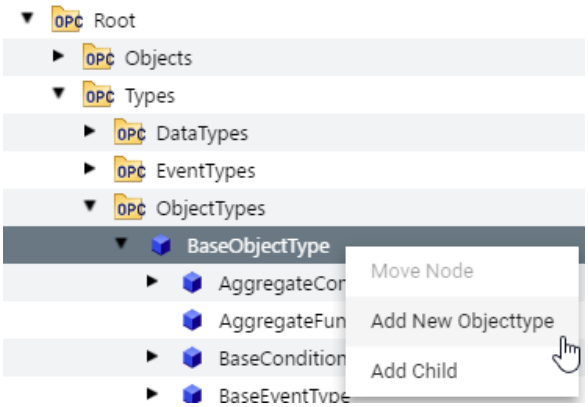

Abbildung 3.5: Erstellen eines Objekttypen

Im Namespace *URI:GenMachineTypeDefV1.0* sind die folgenden Objekttypen als allgemeine Maschinenbeschreibung definiert:

- MachineInformationType (Tabelle 3.1)
- MachineConfigurationType (Tabelle 3.2)
- MachineStatusType (Tabelle 3.3)

Der Namespace *URI:TireProductionV1.0* definiert folgende Objekttypen:

- TireProductionJobType (Tabelle 3.4)
- TireProduction_IMM_MES (Tabelle 3.6)

Es sind im Namespace *URI:TransportV1.0* die folgenden Objekttypen definiert:

- TransportJobType (Tabelle 3.7)
- Transport_IMM_MES (Tabelle 3.10)

Der Namespace *URI:WarehouseV1.0* definiert die folgenden Objekttypen:

- WarehouseJobType (Tabelle 3.11)
- Warehouse_IMM_MES (Tabelle 3.13)

Es wurden somit alle beschriebenen Objekttypen in den jeweiligen *Namespaces* angelegt. Die einzelnen Elemente wurden dann über Rechtsklick auf den Objekttyp unter *AddChild* hinzugefügt. Alle Eigenschaften der Kinder, wie z. B. Name, NodeClass, TypeDefinition oder ReferenceType, wurden beim Hinzufügen editiert.

Erstellen einer Struktur

Einige Kinder der beschriebenen Objekttypen sind Variablen vom Datentyp einer selbst erstellten Struktur. Eigene Strukturen werden als Subtyp des abstrakten Datentyps *Structure* angelegt. Der Node ist unter dem Browse-Path Root/Types/DataTyes/BaseDataTypes/Structure zu finden. Über Rechtsklick auf *Structure* unter *Add New Structured Data Type* wird eine neue anwenderspezifische Struktur erstellt. Neue Elemente können dann durch Rechtsklick auf der bereits erstellten Struktur hinzugefügt werden.

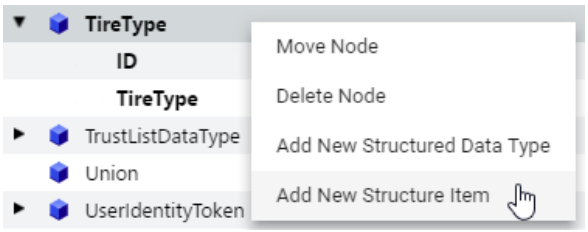

Abbildung 3.6: Hinzufügen von Strukturelementen

Eine erstellte Struktur kann nun als Datentyp eines *Value* Attributes einer Variable verwendet werden. Es wurden damit die beschriebenen Strukturen (Tabelle 3.8) erstellt. Da eine anwenderspezifische Struktur nicht durch die Basisdatentypen beschrieben ist, wird die neu angelegte Struktur über das Typ Dictionary veröffentlicht.

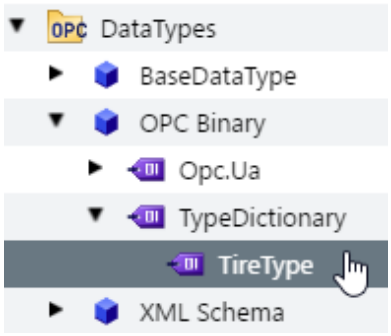

Abbildung 3.7: Typ Dictionary einer anwenderspezifischen Struktur

Im Namespace *URI:GenMachineTypeDefV1.0* sind die folgenden Strukturen definiert:

- TireType (Tabelle 3.8)
- ActiveTransportJobType (Tabelle 3.8)

Es wird die dritte Möglichkeit zur Darstellung von Strukturen (Kapitel 2.2.3 - Datentypen) angewandt. Deshalb muss ebenfalls ein Variablentyp, welcher die Struktur durch Referenzierungen einfacher Variablen abbildet, erstellt werden. Neue Variablentypen werden als Subtyp des Variablentyps *BaseDataVariableType* angelegt. Der Node ist unter dem Browse-Path Root/Types/VariableTyes/BaseVariableTypes/BaseDataVariableTypes zu finden. Über Rechtsklick auf den Node *BaseDataVariableTypes* unter *Add New Variable Type* wird ein neuer Variablentyp erstellt. Variablentypen können lediglich die NodeClass *Variable* referenzieren, da ein Variablentyp nur zur Beschreibung der Bedeutung einer Variable erstellt wird. Es werden nun alle Elemente der Struktur als Kind des Variablentyps referenziert.

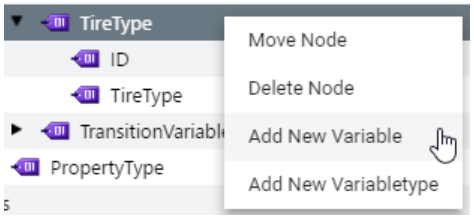

Abbildung 3.8: Variablentyp einer anwenderspezifischen Struktur

Somit wurden alle Strukturen ebenfalls als Variablentyp abgebildet. Der Namespace *URI:GenMachineTypeDefV1.0* definiert die folgenden Variablentypen:

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition
		AcitveTransportJobType		
HasComponent	Variable	AimJobID	String	BaseDataVariableType
HasComponent	Object	TireMaterial	TireType	TireType
		TireType		
HasComponent	Variable	ID	String	BaseDataVariableType
HasComponent	Variable	TireType	Byte	BaseDataVariableType

Tabelle 3.14: Variablentypen

Die Elemente der Struktur einer Steuerungsvariable werden dann beim Mapping auf die Kinder des Variablentyps gemappt. Nun kann auf einzelne Elemente der anwenderspezifischen Strukturen AcitveTransportJobType und TireType über die Kinder des Variablentyps zugegriffen werden. Die gesamte Struktur kann weiterhin über das *Value* Attribut der Variable ausgelesen werden. Über die Referenz *HasTypeDefinition* können die Variablentypen als *TypeDefinition* einer Variable verwendet werden.

Instanziieren eines Objekttypes

Nachdem alle Objekttypen, Datentypen und Variablentypen definiert worden sind, müssen die Schnittstellen noch als Objekt instanziiert werden. Objekte werden mit der Referenz *Organizes* vom *Node Objects* instanziiert. Der *Node* ist unter dem Browse-Path Root/Objects zu finden. Über Rechtsklick auf die *Node* können neue Instanzen von Objekten hinzugefügt werden.

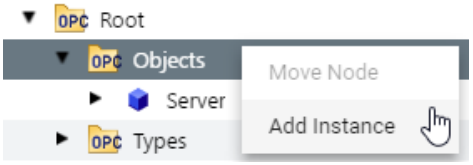

Abbildung 3.9: Instanz eines Objekttyps erstellen

Die Instanzen der jeweiligen Objekttypen wurden in den entsprechenden Namespaces hinzugefügt. Nun sind die Dienste und Informationen auf der Architekturschicht *Information* des RAMI 4.0 beschrieben. Über die Exportfunktion im *Information model* Fenster wurden die drei Informationsmodellierungen für die Teilprozesse in eine XML-Datei exportiert. Da lediglich eine instanziierte Objektstruktur vorliegt, müssen für das Bereitstellen von Informationen und Diensten auf einem OPC UA Server die Elemente der Objekte mit Elementen der

SIMATIC S7-1500 Steuerung verknüpft werden. Hierfür muss ein Steuerungsprogramm entwickelt werden, das die erforderlichen Informationen und Dienste des erstellten Informationsmodells zur Verfügung stellen kann. Bereits bestehende Programme müssen dahingehend angepasst werden.

3.3 OPC UA Server

Der OPC UA Server läuft auf der im Stückgutprozess verwendeten SPS. Es ist eine SIMATIC S7-1516 Steuerung der Firma Siemens AG im Einsatz. Der gesamte Stückgutprozess ist durch diese Steuerung automatisiert. Würde jeder Teilprozess durch eine SPS bzw. durch einen Server dargestellt sein, so würde die Kommunikation zwischen den Komponenten über OPC UA durch die beschriebenen Informationsmodelle stattfinden. Da dies nicht der Fall ist, findet die Kommunikation zwischen den Komponenten lokal statt, jedoch geschieht dies über dieselben Dienste und Informationen. Der OPC UA Server der SPS muss vor dem Beginn der Projektierung in den Hardwareeigenschaften der Steuerung unter OPC UA/Server/Allgemein aktiviert werden.

Nachdem das Projekt neu übersetzt und auf die SPS geladen wurde, kann sich ein Client mit dem OPC UA Server der Steuerung verbinden. Dieser veröffentlicht alle Dienste und Informationen in einem generischen[3] Modell. Wenn ein Informationsmodell von einem OPC UA Server bereit gestellt werden soll, muss vor Beginn der Projektierung geprüft werden, ob der OPC UA Server die benötigten *Services* wie z. B. den Methoden Aufruf bereitstellt. Die Einstellung für die Sichtbarkeit von Variablen oder Methoden kann im Steuerungsprogramm konfiguriert werden. Mithilfe des generischen Modells werden Informationen und Dienste unterschiedlicher Steuerungsprogramme von SIMATIC S7-1500 Steuerungen an dem internen OPC UA Server durch das gleiche Modell beschrieben. Um Informationen und Dienste einer bestimmten Aufgabe oder einer Komponente auf einheitliche Art und Weise zugänglich zu machen, können Informationsmodelle definiert werden. Das generische Modell der SIMATIC S7-1500 kann in den Hardwareeinstellungen unter OPC UA/Server/Allgemein/Standard-Server-Schnittstelle deaktiviert werden. Wird diese Einstellung vorgenommen, so werden nur noch die eingebundenen Informationsmodelle vom OPC UA Server veröffentlicht. Dadurch kann auch ein eingeschränkter Zugriff auf Informationen und Dienste realisiert werden. Die in Kapitel 3.2 definierten Informationsmodelle für das Warenlager, das Transportsystem und die Reifenproduktionsmaschine werden in den OPC UA Server der SPS eingebunden.

[3] „etwas, das sich [allgemein] auf alle Objekte einer Klasse und nicht nur [spezifisch] auf ein einzelnes Objekt dieser Klasse bezieht oder darauf anwenden lässt“, Quelle: http://www.awb1.ch/dat/g/generisch.php, Zugriff: 15.04.2018

3.3.1 Steuerungsprogramm

Das Steuerungsprogramm wurde mit der Software STEP 7 im TIA Portal der Version 15 entwickelt. Die Funktion des Einbindens eines Informationsmodells als OPC UA Server-Schnittstelle wird seit dem Release der Version 15 des TIA Portals im Dezember 2017 unterstützt. Ab der Firmware-Version 2.5 einer SIMATIC S7-1500 wird diese Funktion ebenfalls von der Steuerung unterstützt.

Die Sensorik und Aktorik des Labormodells ist mit einer dezentralen Peripherie verbunden. Diese ist über PROFIBUS DP mit der SPS verbunden. Nachdem ein neues TIA V15 Projekt erstellt worden ist, wurde die Hardware der dezentralen Peripherie sowie der SPS unter „Geräte und Netze“ abgebildet. Die Konfiguration der dezentralen Peripherie sowie der SPS ist im Projektarchiv im Anhang A.4 hinterlegt.

Da kein bestehendes Programm vorliegt, wurde das Steuerungsprogramm neu entwickelt. Die bereits beschriebenen Methoden und Variablen des Informationsmodells (Tabelle 3.1 bis 3.13) müssen in der Programmstruktur enthalten sein. Die Programme wurden, genau wie auch die Teilprozesse, in drei einzelne Programmstrukturen Warenlager, Transportsystem und Reifenproduktionsmaschine unterteilt. Eine Programmstruktur implementiert ein Prozess-Steuerungsprogramm, eine Auftragsverwaltung und das Handling der OPC UA Methoden. Die gesamte Programmstruktur befindet sich im Anhang A.4 im dort hinterlegten Projektarchiv.

OPC UA Methoden in der SPS

Für das Bereitstellen einer OPC UA Methode am Server stehen zwei Systemfunktionen zur Verfügung. Werden die Systemfunktionen *OPC_UA_ServerMethodPre* und *OPC_UA_ServerMethodPost* in einem Funktionsbaustein aufgerufen, so stellt der OPC UA Server der SPS eine Methode unter dem Namen des Instanzen-DB des Funktionsbausteines bereit. Der Programmablauf zum Bereitstellen einer OPC UA Methode ist durch ein UML-State-Chart modelliert, welches im Anhang A.1 unter der Abbildung A.1 zu finden ist. Durch das Bereitstellen von OPC UA Methoden über die Systemfunktionen wird eine Kapselung von lokalen Funktionen und OPC UA Methoden erreicht. Über dieses Handling können auch bereits bestehende Funktionen mit eventuell anderen Parametern, als die durch ein Informationsmodell beschriebenen Methoden, durch einen Wrapper[4] aufgerufen werden.

[4] „Der Begriff Wrapper (abgeleitet von "to wrap up": einwickeln) wird im Software-Engineering in verschiedenen Zusammenhängen verwendet, und lässt sich dabei aber grundsätzlich als eine Software - die wiederum mindestens eine oder auch mehrere Software-Komponenten umhüllt - definieren.“, Quelle: https://www.itwissen.info/Wrapper-wrapper.html, Zugriff: 11.04.2018

Die Kapselung von lokalen Funktionen und OPC UA Methoden ist jedoch aufwendiger als ein direktes Bereitstellen von SPS Funktionen.

Durch den Aufruf der Systemfunktion *OPC_UA_ServerMethodPre* wird der Methodenaufruf am OPC UA Server gesteuert. Wird eine OPC UA Methode am Server durch einen Client aufgerufen, so meldet die Funktion *OPC_UA_ServerMethodPre* über die Variable *Called* den Aufruf. Eventuelle Eingabeparameter werden durch eine entsprechende Struktur dem Funktionsbaustein übergeben. Nun kann ein lokaler Aufruf einer Funktion erfolgen. Ist der Aufruf der gewünschten Funktionen abgeschlossen, so wird die Variable *Finished* der Systemfunktion *OPC_UA_ServerMethodPost* gesetzt. Eventuelle Ausgabeparameter werden über eine Struktur der OPC UA Methode zurückgeliefert. Wird eine Methode durch einen Client am Server aufgerufen, so kann während des Aufrufs kein weiterer Client diese Methode am Server nutzen, bis wieder der Status *PreMethod* des UML-State-Chart (Anhang A.1 - Abb. A.1) erreicht ist. Jeder Dienst, der durch das bereits entwickelte Informationsmodell (Kapitel 3.2) beschrieben worden ist, wird durch den UML-State-Chart modellierten Programmablauf (Anhang A.1 - Abb. A.1) am OPC UA Server bereitgestellt.

Prozesssteuerung

Die Übersicht der Bausteine des Steuerungsprogramms ist im Anhang A.1 Tabelle A.1 bis A.3 aufgelistet. Alle Instanz-Datenbausteine wurden für eine bessere Übersicht weggelassen. Für jeden Teilprozess wird ein eigener Organisationsbaustein erstellt, der die Funktionen für das Prozess-Steuerungsprogramm und die Funktionen für das Bereitstellen der OPC UA Methoden aufruft. Die Prozesssteuerung der Reifenproduktionsmaschine automatisiert das Ansteuern des Laufbandes und des Drehschiebers. Jedes Stückgut wird durch eine Marke in einem Petrinetz symbolisiert. Die Informationen zu einem Stückgut an einer Position auf dem Laufband, wie Auftrags- und Materialinformationen, werden durch die Prozesssteuerung in einem Array verwaltet. Wenn ein Stückgut nach einer Übergabeanfrage durch das Transportsystem auf das Laufband abgelegt wird, so wird in das Petrinetz eine neue Marke hinzugefügt und die Stückgut-Informationen werden aus der Übergabeanfrage in die Prozessverwaltung übernommen. Da ein Petrinetz diese Parallelität bietet, wurde das Programm mit dieser Beschreibungsart modelliert (Anhang A.1 - Abb. A.2).

Die Prozesssteuerung des Warenlagers automatisiert den Ausschub eines Stückguts. Das Programm wurde durch ein UML-State-Chart modelliert (Anhang A.1 - Abb. A.4). Fordert das Transportsystem erfolgreich einen neuen Baustein an, so wird der Prozess gestartet. Die Prozesssteuerung des Transportsystems automatisiert den Transportarm und die Kommunikation zu dem Warenlager und der Reifenproduktionsmaschine. Das Programm wurde durch ein UML-State-Chart modelliert (Anhang A.1 - Abb. A.3). Sobald Transportaufträge

vorliegen, erteilt die Prozesssteuerung dem Warenlager Ausgabeaufträge. Das Transportgut wird dann zur Reifenproduktionsmaschine transportiert. Nach erfolgreicher Anfrage der Übergabe wird das Transportgut abgelegt. Der Transportarm fährt nun wieder in seine initialisierte Position.

Auftragsverwaltung

Die Übersicht der Bausteine der Auftragsverwaltung aller Teilsysteme ist im Anhang A.1 Tabelle A.1 bis A.3 aufgelistet. Alle Funktionen der Auftragsverwaltungen werden als OPC UA Methoden bereitgestellt. Da die Kommunikation zwischen dem Warenlager und dem Transportsystem sowie zwischen der Reifenproduktionsmaschine und dem Transportsystem lokal ist, werden die Funktionen der Auftragsverwaltung in den lokalen Prozesssteuerungen direkt aufgerufen. Würde jeder Teilprozess mit einer eigenen SPS automatisiert, so würde der Aufruf in den Prozesssteuerungen über OPC UA Methoden anstatt lokaler Funktionen stattfinden. Die Architektur des Programms würde sich jedoch nicht ändern. Jeder Teilprozess hat eine eigene Auftragsverwaltung. Ausgabeaufträge des Warenlagers werden angenommen, sobald mindestens ein Stückgut eingelagert ist und der Prozess des Auswurfs beendet wurde. Das Warenlager nimmt Einlagerungsaufträge an, sofern nicht mehr als zehn Stückgüter im Lager vorhanden sind. Bei der Einlagerung wird der Stückgut-Typ und eine ID des Stückguts übergeben und gespeichert.

Das Transportsystem kann bis zu zehn Transportaufträge entgegennehmen. Alle weiteren Transportaufträge werden abgelehnt. Sobald ein Transportauftrag beendet wurde, können wieder neue Aufträge angenommen werden. Ein Transportauftrag beinhaltet eine Job-ID, eine Stückzahl, den Stückgut-Typ und ein Transportziel. Bei einem Ablageantrag an die Reifenproduktionsmaschine wird der aktive Transportauftrag übergeben. Die Reifenproduktionsmaschine nimmt den Ablageantrag an, sowie alle erforderlichen Plätze auf dem Laufband frei sind. An der Reifenproduktionsmaschine können bis zu zehn Reifen-Produktionsaufträge angenommen werden. Ein Reifen-Produktionsauftrag beinhaltet eine Job-ID, eine Stückzahl und den Stückgut-Typ. Die Auftragsverwaltungen nehmen keine Aufträge entgegen, sobald der *SleepMode* der jeweiligen Maschine aktiviert worden ist. Aktive Prozesse werden noch beendet, jedoch können währenddessen keine neuen Aufträge entgegengenommen werden, und zwar so lange, bis der *SleepMode* des jeweiligen Teilprozesses deaktiviert wurde.

3.3.2 Mapping auf ein Steuerungsprogramm

Nachdem das Steuerungsprogramm erstellt wurde, können die Elemente des Informationsmodells auf das Steuerungsprogramm gemappt werden. Dies wird mithilfe des Tools SiOME (Kapitel 3.2.1) realisiert. Die herstellerspezifischen Hinweise für das Einbinden einer

OPC UA Informationsmodellierung an dem internen OPC UA Server ist im Kommunikationshandbuch der SIMATIC Steuerungen beschrieben (Siemens AG, 2017, S. 218-234). Zunächst wird eine erstellte Informationsmodellierung für z. B. die Reifenproduktionsmaschine aus der abgespeicherten XML-Datei importiert. Es wurde bereits eine Instanz des Objekttyps TireProduction_IMM_MES angelegt. Die Nodes des Objektes werden nun auf Elemente des Steuerungsprogramms gemappt. Hierfür wird das TIA Projekt des Steuerungsprogramms über das Fenster TIA Portal geöffnet. Nach dem Öffnen des Projekts kann durch die Struktur navigiert werden.

Alle Bausteine, die für das Mapping benötigt werden, müssen durch das Auswahlfeld aktiviert werden. Danach muss der Schalter Details im Fenster TIA Portal aktiviert werden. Es sind nun alle Details wie Variablen oder Instanzen-DBs sichtbar. Im Fenster Information model muss der Schalter Mapping aktiviert sein. Nun werden Variablen aus dem Projekt per Drag and Drop auf die entsprechenden Nodes gemappt. Bei OPC UA Methoden wird ein Instanzen-DB eines Funktionsbausteins, welcher eine OPC UA Methode veröffentlicht (Anhang A.1 - Abb. A.1), aus dem Projekt per Drag and Drop auf den entsprechenden Node gemappt. Der Programmaufbau beim Mapping ist in folgender Abbildung dargestellt.

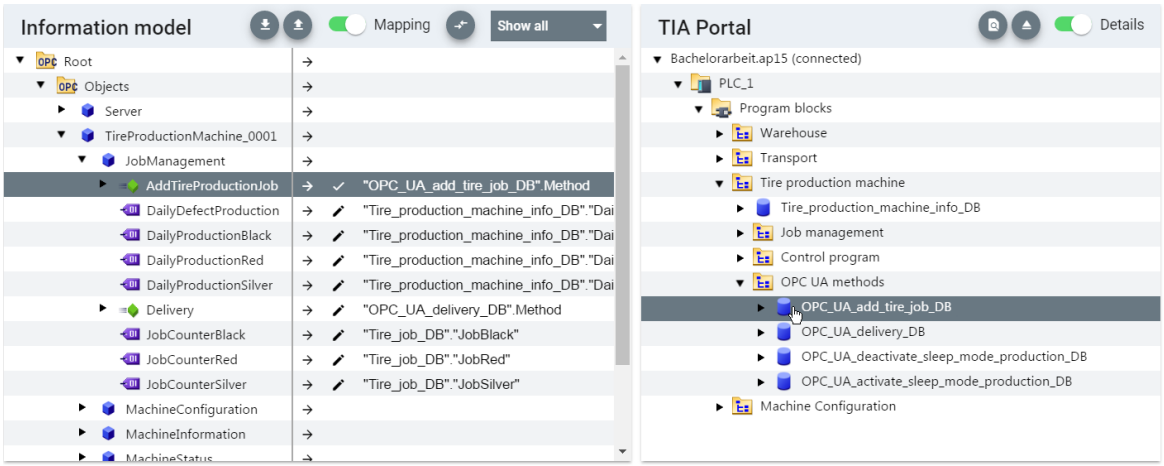

Abbildung 3.10: SiOME Mapping

Nachdem alle Nodes der Objektinstanz auf Steuerungselemente gemappt wurden, ist die fertige Schnittstellenbeschreibung in eine XML-Datei (Anhang A.3) exportiert worden. Das Mapping auf Steuerungs-Variablen und -Methoden in einer XML-Datei ist in Kapitel 3.2.1 beschrieben. Das gemappte Informationsmodell kann nun in den OPC UA Server der SPS eingebunden werden.

3.3.3 Einbinden des Informationsmodells am OPC UA Server

Nachdem der OPC UA Server in den Hardwareeigenschaften der Steuerung aktiviert wurde, können nun die auf das Steuerungsprogramm gemappten Informationsmodelle importiert werden. In der Projektnavigation unter der Rubrik OPC UA-Kommunikation/Server-Schnittstellen/Server-Schnittstelle importieren, werden die XML-Dateien eingebunden. Folgende Dateien sind als Server-Schnittelle importiert worden.

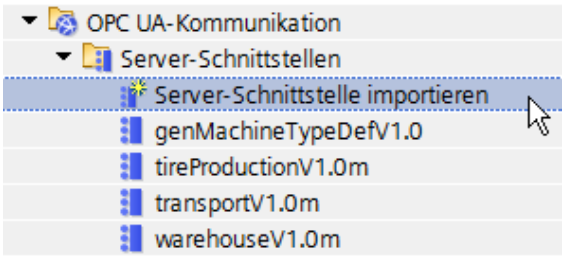

Abbildung 3.11: TIA Portal OPC UA Schnittstellen Import

Beim Einbinden werden im System die Mappings überprüft und intern hinzugefügt. Nach dem Einbinden der XML-Dateien muss das Projekt neu übersetzt und auf die SPS geladen werden. Die Dienste und Informationen sind nun über eine im Vorfeld definierte Schnittstelle über OPC UA zugänglich. Somit ist auch die Schicht *Communication* der Architekturachse des RAMI 4.0 (Abb. 2.1) beschrieben. Für eine ganzheitliche Betrachtung der Schicht *Communication* wird ebenfalls der Kommunikationsteilnehmer des OPC UA Clients untersucht.

3.4 OPC UA Client

(vgl. Mahnke u. a., 2009, S. 20 f.)

OPC UA Clients können verschiedene Aufgaben und Funktionen übernehmen. Diese können z. B. ein SCADA-System, ein HMI oder ein Prozessleitsystem repräsentieren oder als M2M-Kommunikation dienen. Es wird hierbei die Unterscheidung zwischen generischen und spezifischen Clients gemacht. Ein generischer Client ist für keine spezielle Aufgabe entwickelt. Durch Browsen der Nodes wird der gesamte Adressraum des OPC UA Servers abgebildet. Ein generischer Client kann z. B. als Test-Client für einen OPC UA Server in der Entwicklung dienen, wie z. B. der Client UaExpert von Unified Automation. Es können Subscriptions angelegt, Methoden aufgerufen oder Strukturen dekodiert werden. Hierbei erfolgt die Initiierung der Aktionen durch den Benutzer.

Wird ein Client für eine bestimmte Aufgabe entwickelt, so ist dies ein spezifischer OPC UA Client. Dieser ist speziell für eine Instanz eines bestimmten Typs entwickelt und kann somit z. B. auch für die Instanz einer Kommunikationsschnittstelle eines Informationsmodells entwickelt werden. Dadurch können z. B. grafische Verläufe oder eine Übersicht angezeigt werden. Jeder Server, der eine Instanz dieses Typs bereitstellt, kann ohne großen Aufwand ausgetauscht werden.

> „Der häufigste Anwendungsfall der Integration von Gerätedaten [einer Objektinstanz] besteht darin, sie in einem Prozessleitsystem zu aggregieren und über das Prozessleitsystem [..] [einem] Client zur Verfügung zu stellen.“ (Mahnke u. a., 2009, S. 21)

Heute werden OPC UA Clients ebenfalls häufig für die Datenerfassung oder Datensammlung genutzt. Die Bereitstellung von Informationen und Diensten über OPC UA findet heute somit primär zwischen der Steuerungsebene und höheren Hierarchieebenen statt (Abb. 2.2). Der Zugriff auf die Daten eines OPC UA Servers findet weitestgehend über *Data Access* (Kapitel 2.2.2 - Part 8) statt, da generische Clients wie HMI-Systeme oft nur den einfachen Variablenzugriff über *Data Access* unterstützen. Informationsmodelle welche z. B. Strukturen, Methoden oder historische Daten verwenden, können mit den meisten heutigen generischen Clients nicht verwendet werden. Aus diesem Grund wurden die einzelnen Elemente der Strukturen ebenfalls über die Variablentypen veröffentlicht (Kapitel 2.2.3 - Datentypen), um einen Weg für den einfachen Variablenzugriff zu schaffen.

Es wurde ein spezifischer Client für die Interaktion mit den Informationsmodellen der drei Teilsysteme entwickelt, damit die benötigten Spezifikationen vom Client unterstützt werden. Der spezifische Client stellt das MES-System sowie die Zulieferung dar (Abb. 3.2). Durch das MES-System werden Produktionsaufträge an die Reifenproduktionsmaschine gestellt. Ebenfalls soll das MES-System eine Übersicht über die aktuelle Auftragslage der Fabrik geben. Der Zulieferer stellt Einlagerungsaufträge an das Warenlager. Da es sich bei einem MES-System um ein Computerprogramm handelt, das Schnittstellen zu verschiedensten Anwendungen ermöglicht, wurde der spezifische OPC UA Client als eine Windows Anwendung entwickelt.

3.4.1 Anwendungsarchitektur

Die OPC-Foundation stellt für die Implementierung von Server und Client Anwendungen einen kostenlosen UA-Stack bereit. Dieser steht für Java, .NET und ANSI C zur Verfügung. Der UA-Stack beinhaltet Low-Level-Funktionen und ist für das Encodieren und Decodieren von Nachrichten, für transportbezogene Sicherheitsfunktionen sowie für den Transport der

Nachrichten zuständig. Über der UA-Stack Schicht befindet sich die SDK (Software Development Kit)-Schicht. Ein OPC UA-SDK beinhaltet High-Level-Funktionen. Diese managen die im UA-Stack implementierten Services wie z. B. das Erstellen, Aktivieren und das Schließen einer Session. Es werden von verschiedenen Mitgliedern der OPC Foundation SDKs angeboten. Mithilfe eines SDKs werden Server- oder Client-Anwendungen implementiert. Diese befindet sich auf der Application Schicht. Eine Server- oder Client-Anwendung kann z. B. ein Userinterface oder anwendungsspezifische Funktionen implementieren. (vgl. Mahnke u. a., 2009, S.255-264) Die Übersicht der beschriebenen Architektur ist in folgendem Abbild dargestellt.

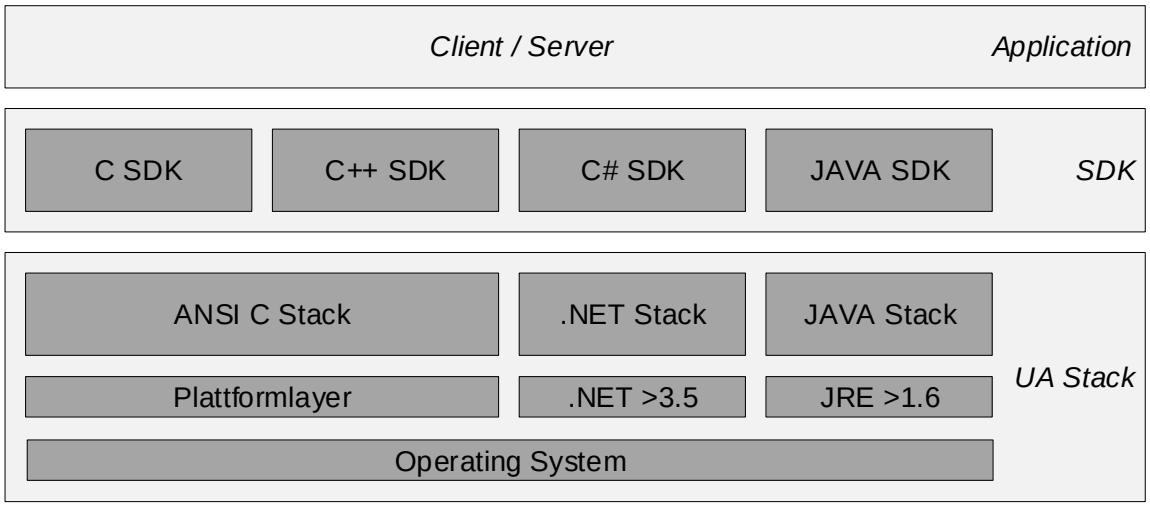

Abbildung 3.12: Anwendungsarchitektur OPC UA (nach ascolab, 2018a)

Für die Entwicklung des spezifischen Clients wird ein SDK benötigt. Es gibt unterschiedliche kommerzielle Lösungen wie z. B. das SDK von Unified Automation oder von Softing. Jedoch gibt es auch verschiedene kostenlose Lösungen. Die Siemens AG bietet im Rahmen eines kostenlosen Anwendungsbeispiels eine Grundlage zur Entwicklung eines OPC UA Clients. Da dieses Anwendungsbeispiel alle benötigten Anforderungen erfüllt, wird der spezifische OPC UA Client auf Grundlage dessen entwickelt. Dieses kann auf den *Siemens Industry Online Support* Seiten unter der Beitrags-ID 109737901 (siehe Siemens AG, 2018a) geladen werden.

Das Visual Studio Projekt des Anwendungsbeispiels beinhaltet alle drei Schichten der Anwendungsarchitektur (Abb. 3.12). Es wird der .NET Stack der OPC-Foundation verwendet. Die SDK Schicht sowie die Anwendungsschicht werden in C# entwickelt. Alle Funktionen der SDK Schicht werden in der Klasse *UAClientHelperAPI* des bestehenden Anwendungsbeispiels implementiert. Diese wurde um weitere Funktionen zum Lesen von Namespaces

erweitert. Das Klassendiagramm der Klasse *UAClientHelperAPI* ist im Anhang A.1 unter der Abbildung A.5 dargestellt. Die Beschreibung der Methoden der Klasse ist in der Dokumentation des Anwendungsbeispieles hinterlegt (siehe Siemens AG, 2018a).

3.4.2 Spezifischer OPC UA Client

Auf der Anwendungsschicht (Abb. 3.12) wurde der spezifische OPC UA Client entwickelt. Dieser stellt das MES-System sowie den Zulieferer dar. Der spezifische Client ist als Visual Studio Projekt im Anhang A.5 hinterlegt. Die Client Anwendung wurde im Rahmen dieser Arbeit in der Klasse *UAClientForm* implementiert. Diese erbt von der Klasse *System.Windows.Forms.Form* und implementiert die grafische Oberfläche. Die Anwendung ist in sieben Tabs gegliedert: *Server*, *Namespaces*, *Factory Overview*, *Warehouse*, *Transport*, *Tire Production* und *Browse Nodes*. Der Client implementiert die Kommunikation nach dem Interaktionsmodell (Abb. 2.3). Die einzelnen Schritte des Interaktionsmodells sowie die Bedienung über die einzelnen Tabs werden im Folgenden für den spezifischen Client genauer beschrieben.

Security vereinbaren

Als erster Schritt wird im Interaktionsmodell die Sicherheit der Kommunikationsverbindung vereinbart. Für die Kommunikation mittels OPC UA bedeutet dies, dass die Verbindung zu einem gewählten *Endpoint* (Kapitel 2.2.2 - Part 2), der einen bestimmten Sicherheitsstandard bereitstellt, aufgebaut wird. In der grafischen Oberfläche des spezifischen Client kann dieser Schritt vom Benutzer im Tab *Server* initiiert werden. Das folgende Squenzdiagramm zeigt den Programmablauf für die Wahl des *Endpoints* sowie für den Verbindungsaufbau.

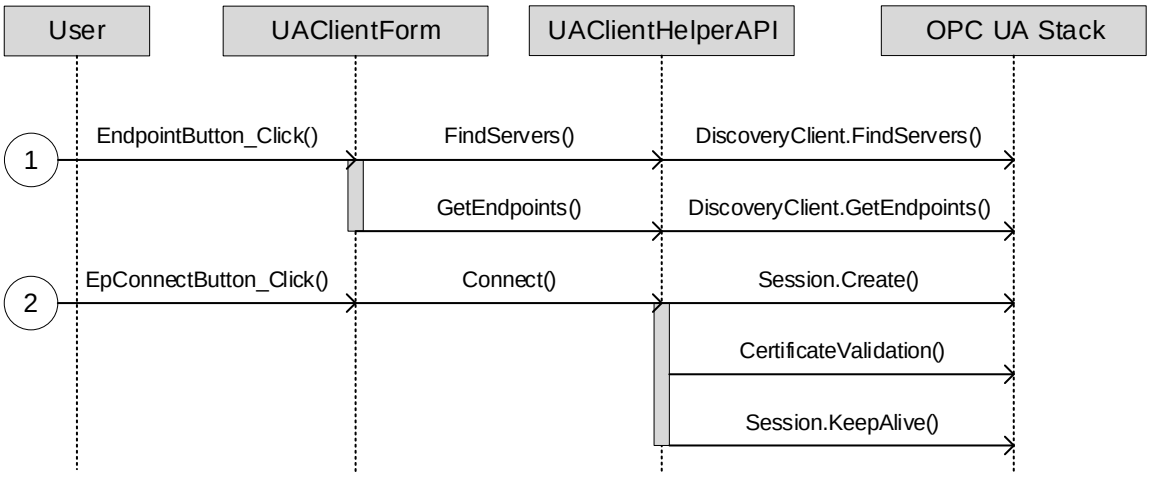

Abbildung 3.13: Sequenzdiagramm: Security vereinbaren

Identität austauschen

Nachdem die Sicherheit der Kommunikation gewählt wurde und eine Kommunikation aufgebaut ist, müssen vor dem Beauftragen einer Aufgabe die Identitäten ausgetauscht werden. In der grafischen Oberfläche des spezifischen Client kann dieser Schritt vom Benutzer im Tab *Namespaces* initiiert werden. Für den Aufbau des Labormodells wird lediglich die Identität der Maschine bzw. des Servers für die Auftragsverwaltung benötigt. Die Feststellung der Identität des Clients durch z. B. eine Benutzeranmeldung am Server wird in diesem Modell vernachlässigt. Für eine reale Produktion ist die Feststellung der Identität des Clients nicht zu vernachlässigen. Die Identität der bereitgestellten Objekte des Servers ist durch das bereits erstellte Informationsmodell gegeben.

Alle drei Teilsysteme wurden durch eine Objektstruktur in einem eigenen *Namespace* am Server beschrieben. Die im Vorfeld definierten Objekttypen sowie der definierte *Namespace* dienen als Identifikation der Maschine. Für die Teilsysteme wurde eine Klasse im Client erstellt, welche Funktionen und Eigenschaften bereitstellt, um ein Objekt am Server zu identifizieren und alle *Node-IDs* von diesem zu laden. Hierfür wurde die Klasse GenMachineTypeDef (Anhang A.2 - Abb. A.6) im *Namespace* TireProductionObjectTypes der OPC UA Client Anwendung implementiert. Es wurden ebenfalls Klassen für alle drei Teilsysteme definiert. Diese erben von der Klasse GenMachineTypeDef. Im Konstruktor der jeweiligen Klasse wird die *Namespace-URI* sowie der Objekttyp des Objektes am OPC UA Server definiert. Das folgende Sequenzdiagramm zeigt den Programmablauf für die Identifikation einer Maschine, die durch das jeweilige Informationsmodell beschrieben ist.

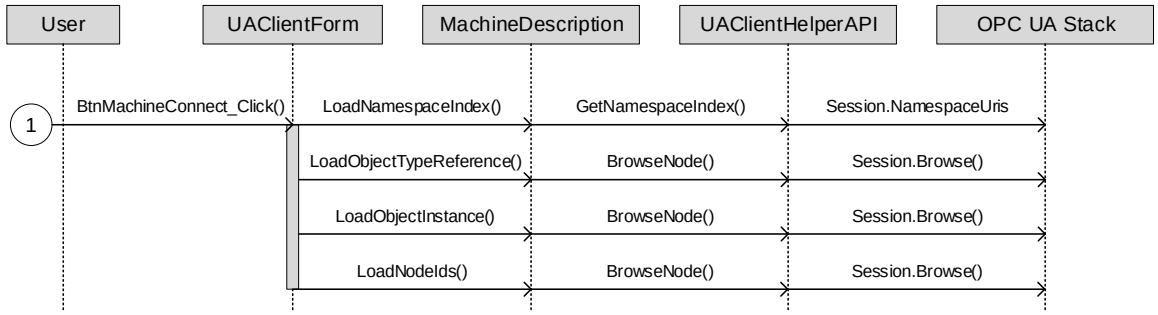

Abbildung 3.14: Sequenzdiagramm: Identität austauschen

Zu Beginn wird festgestellt, ob der jeweilige *Namespace* am Server bereitgestellt ist. Wenn dies der Fall ist, so wird die *Node-ID* des zuvor definierten Objekttyps geladen. Nachdem diese bekannt ist, wird im *Objects* Ordner nach einer Instanz dieses Objekttyps gesucht. Dadurch ist eine eindeutige Identifizierung eines Objektes möglich. Die BrowseNode Methode der Klasse UAClientHelperAPI gibt eine ReferenceDescriptionCollection zurück. Diese enthält alle referenzierten *Nodes* eines *Source Nodes*. Alle *Node-IDs* einer ReferenceDescriptionCollection werden nach erfolgreichem Ausführen der Funktion LoadNodeIds in Listen der Klasse GenMachineTypeDef gespeichert. Über dieses Verfahren werden die *Node-IDs* aller Kinder eines Objektes geladen und gespeichert. Die Zuordnung der *Node-ID-Collections* wird über im *Namespace* TireProductionObjectTypes global definierte Enums (Anhang A.2 - Abb. A.7) erreicht. Die Zahl eines Enum Elements entspricht der Position der *Node-ID* in der jeweiligen Liste. Durch die im Vorfeld zugeordneten *Node-IDs* kann der spezifische Client offline gezielt *Nodes* verwenden.

Initiierung und Beauftragung einer Aufgabe

Nachdem die Maschine über den *Namespace* und den Objekttyp identifiziert wurde und alle *Node-IDs* der Kinder gespeichert und mit Elementen des spezifischen Clients verknüpft sind, können Aufträge an die Maschine initiiert werden. Das folgende Squenzdiagramm zeigt den Programmablauf für die Initiierung und Beauftragung einer Aufgabe.

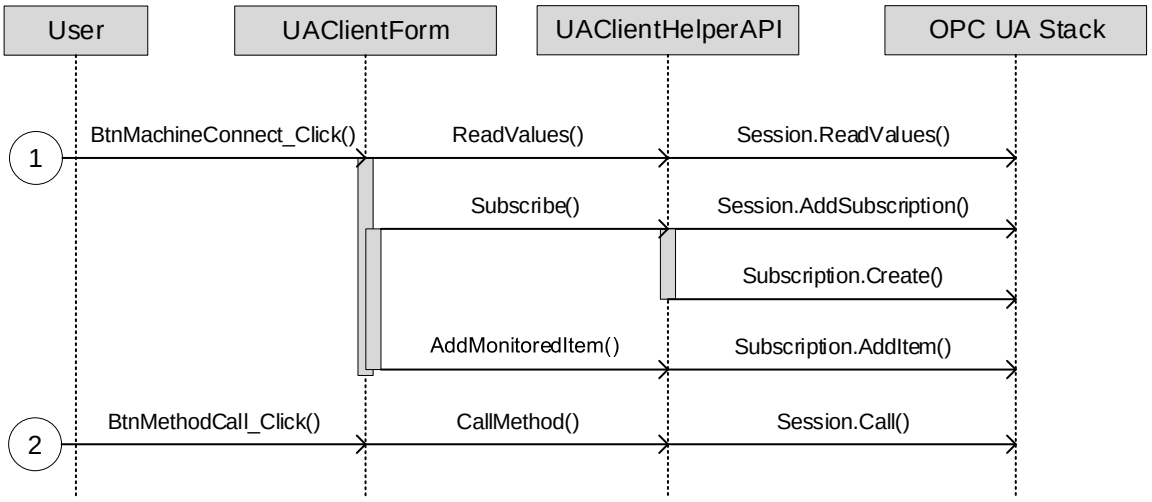

Abbildung 3.15: Sequenzdiagramm: Initiierung und Beauftragung einer Aufgabe

Zu Beginn werden die Maschineninformationen einmalig gelesen. Alle Variablen, die sich während des Prozesses ändern, werden durch eine *Subscription* abonniert. Es können mehrere Variablen zu einer *Subscription* als *MonitoredItem* hinzugefügt werden. Ändert sich das *Value* Attribut einer abonnierten Variable, so wird der aktuelle Wert über einen *EventHandler* veröffentlicht. Die zwei Schritte Lesen und Abonnieren werden automatisch nach erfolgreicher Identifizierung der Maschine vorgenommen. Nachdem alle *Subscriptions* angelegt wurden, werden die aktuellen Werte der Fabrikübersicht im Tab *Factory Overview* der Anlage angezeigt. Der Tab *Factory Overview* ist in folgendem Abbild dargestellt.

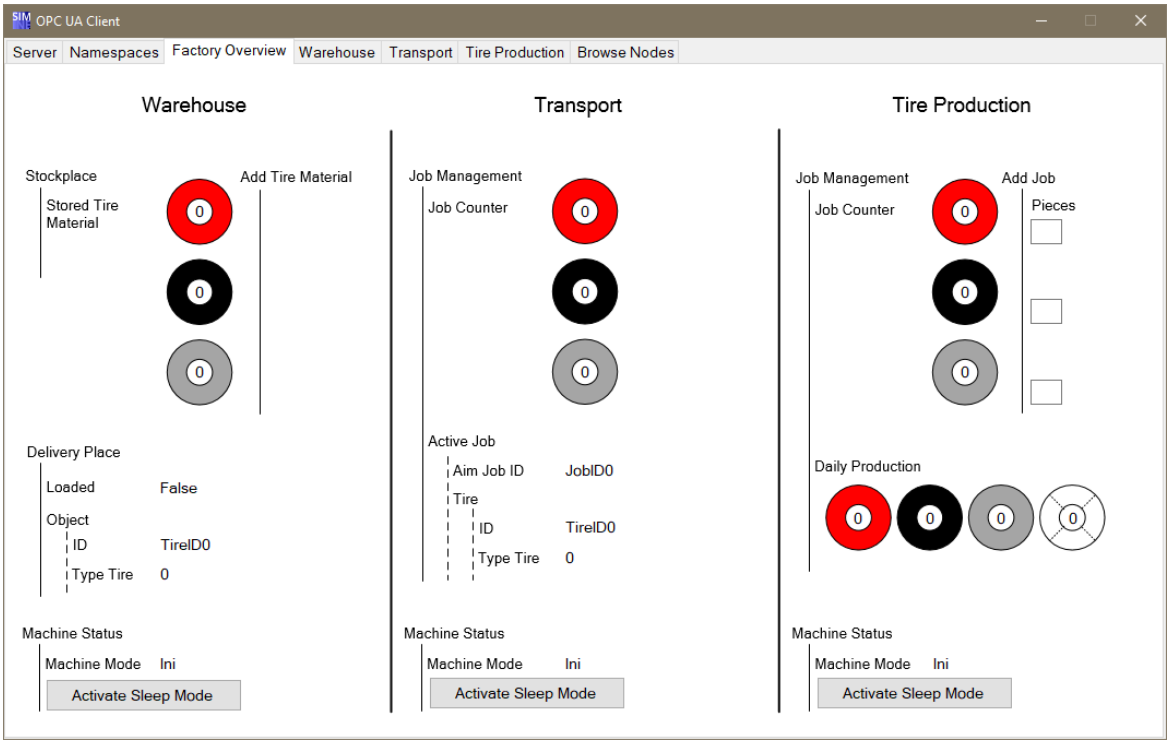

Abbildung 3.16: OPC UA Client - Tab Factory Overview

Die Fabrikübersicht ist in die drei Teilsysteme gegliedert. Im Warenlager können Einlagerungsaufträge durch das Klicken in die jeweiligen Stückgüter gestellt werden. Der Gesamtzähler der einzelnen Stückguttypen ist in den jeweiligen Stückgütern dargestellt. In allen Teilsystemen kann der *SleepMode* aktiviert und deaktiviert werden. Das Transportsystem wird nicht durch das MES-System angesteuert. Somit werden nur die aktuellen Informationen des Transportsystems dargestellt. Durch das Klicken auf die jeweiligen Stückgüter der Reifenproduktionsmaschine können Produktionsaufträge gestellt werden. Zuvor muss die jeweilige Stückzahl des Produktionsauftrags in das Eingabefeld eingetragen werden. Vor dem erneuten Start der Anwendung kann ausgewählt werden, ob der zuletzt verbundene Server automatisch wieder verbunden werden soll.

Durchführung einer Aufgabe

Die gestellten Aufträge werden durch das Steuerungsprogramm (Kapitel 3.3) ausgeführt. Der lesende Zugriff auf Informationen durch Subscriptions oder durch den Datenzugriff *Data Access* ist lediglich zum Darstellen von Informationen am spezifischen Client eingesetzt. Das Anfragen und Erteilen von Aufträgen wurde über OPC UA Methoden realisiert.

Beenden einer Aufgabe

Nachdem alle Aufgaben einer Kommunikationsbeziehung beendet sind, kann die Kommunikation zwischen den zwei Teilnehmern ebenfalls beendet werden. Dies ist z. B. bei Wartungen oder Austausch einer Komponente der Fall. Beim Beenden einer OPC UA Kommunikationsbeziehung wird die beim Verbindungsaufbau erstellte *Session* geschlossen. Das Schließen der *Session* wird ebenfalls automatisch eingeleitet, wenn der Server keine *Keepalive-Notifications* mehr sendet. Das folgende Squenzdiagramm zeigt den Programmablauf für das manuelle Beenden der Kommunikationsbeziehung.

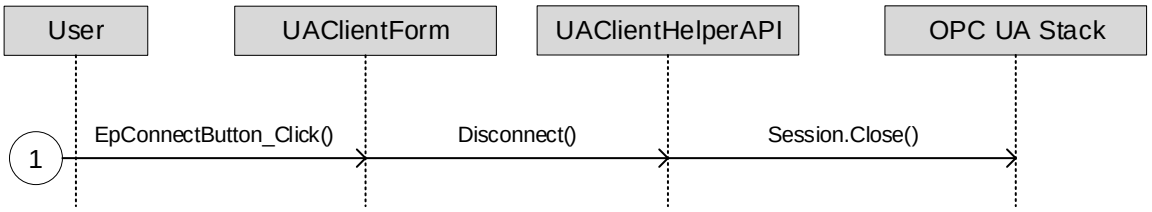

Abbildung 3.17: Sequenzdiagramm: Beenden einer Aufgabe

4 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war es, eine Umsetzung einer standardisierten Kommunikationsschnittstelle mittels OPC UA aufzuzeigen. Ebenfalls war es das Ziel, die Integration von Komponenten durch eine standardisierte Kommunikationsschnittstelle zu vereinfachen.

Es wurde eine Informationsmodellierung für das Labormodell eines Stückgutprozesses erstellt. Diese wurde auf Grundlage der für Industrie 4.0 beschriebenen Kommunikationskonzepte entwickelt. Die Schnittstelle für das Warenlager, das Transportsystem sowie der Reifenproduktionsmaschine wurde als Objekttyp abgebildet und am Server als Instanz bereitgestellt. Das Informationsmodell ist herstellerunabhängig und wurde in einer XML Datei abgebildet. Diese kann von allen Herstellern verwendet werden, welche die jeweiligen OPC UA Spezifikationen auf ihren Komponenten implementiert haben.

Das Steuerungsprogramm der SPS wurde auf Basis der Informationen und Dienste des Informationsmodells neu entwickelt. Vor dem Einbinden des Informationsmodells am OPC UA Server der SPS wurden die Elemente des Informationsmodells mit den Elementen des Steuerungsprogramms verknüpft. Alle OPC UA Server, welche die Instanz des jeweiligen Objekttyps des Informationsmodells veröffentlichen, bilden durch den selben Objekttyp ebenfalls die selbe Struktur ab. Dadurch sind die Informationen und Dienste über eine standardisierte Kommunikationsschnittstelle zugänglich.

Für den Zugriff auf die standardisierten Informationen und Dienste wurde ein spezifischer OPC UA Client entwickelt. Dieser interagiert mit dem OPC UA Server anhand des Interaktionsmodells für Industrie 4.0-Komponenten. Beim Schritt „Identität austauschen“ lädt der spezifische OPC UA Client alle aktuellen Node-IDs der Objektinstanz der Kommunikationsschnittstelle. Dadurch ist das Konzept von Plug-and-Work möglich, da alle OPC UA Server, welche eine Instanz des jeweiligen Objekttyps bereitstellen, mit dem spezifischen Client interagieren können. Es konnte erfolgreich die Umsetzung einer standardisierten Kommunikationsschnittstelle aufgezeigt werden. Ebenfalls wurde dadurch die Integration einer Komponente vereinfacht. Die aufgezeigte Umsetzung kann als Leitfaden genutzt werden.

Bei der Umsetzung der Informationsmodellierung wurde schnell deutlich, dass ein fundiertes Grundlagenwissen von OPC UA für die Entwicklung dieser benötigt wird. Die Definition der Informationsmodellierung setzt ebenfalls eine detaillierte Prozesskenntnis voraus. Branchenweite standardisierte Informationsmodelle werden häufig durch Industrieverbände

in Kooperation mit der OPC-Foundation entwickelt. Die Definition dieser Companion Specifications kann über Jahre dauern. Der Aufwand für die Definition und Umsetzung eines branchenweiten Standards durch ein Informationsmodell ist somit sehr hoch. Für den jeweiligen Anwendungsfall sollte eine Beurteilung des effektiven Nutzens einer standardisierten Kommunikationsschnittstelle getroffen werden. Ebenfalls sollte beurteilt werden, ob für einen Anwendungsfall überhaupt ein allgemeiner Standard entwickelt werden kann. Jedoch kann ein Informationsmodell auch als einfach definierte Schnittstellenbeschreibung für kleine Anwendungen oder als Einschränkung des Zugriffs von Informationen und Diensten verwendet werden.

Werden Informationsmodelle an einem OPC UA Server einer SPS bereitgestellt, so ist die Entwicklung des Steuerungsprogramms ein entscheidender Faktor für die Beurteilung. Neu entwickelte Steuerungsprogramme können auf aktuelle Companion Specifications angepasst werden. Oft werden jedoch Teile von bestehenden Steuerungsprogrammen wieder verwendet. Dadurch kann es dazu kommen, dass ein Informationsmodell möglicherweise nicht kompatibel zu bestehenden Steuerungsprogrammen ist. Es müssten dann die entsprechenden Wrapper-Funktionen geschrieben werden, welche den Diensten und Informationen der Informationsmodellierung entsprechen. Die Machbarkeit und der Aufwand sollte für jeden Anwendungsfall neu beurteilt werden.

Die momentan größte Herausforderung stellen jedoch die generischen OPC UA Clients in z.B. einem HMI-System, einem SCADA-System, einem Prozessleitsystem oder einer SPS dar. Diese unterstützen oft lediglich den Zugriffstyp *Data Access*. Stellt ein OPC UA Server ein Informationsmodell bereit, welches z.B. Methoden oder historische Daten verwendet, so kann diese heute von den meisten generischen OPC UA Clients nicht vollständig verwendet werden. In der Praxis werden häufig generische OPC UA Clients verwendet, um Informationen von einen OPC UA Server zu beziehen. Dadurch kann die Realisierung einer standardisierten Kommunikationsschnittstelle durch ein Informationsmodell in der Praxis noch nicht voll umfänglich umgesetzt werden. Aufgrund dessen wurde im Rahmen dieser Arbeit ein spezifischer Client entwickelt, der alle nötigen Spezifikationen unterstützt. Die Entwicklung der generischen OPC UA Clients könnten in der Zukunft jedoch weiter fortgeschritten sein, sodass dann Methoden oder historische Daten eines Informationsmodells unterstützt werden. Denkbar wäre auch, dass zukünftig durch das Laden eines Informationsmodells in den generischen Client, ähnlich wie bei dem entwickelten spezifischen Client, die aktuellen Node-IDs der Objektinstanz vom OPC UA Server automatisch geladen werden können. Ebenfalls würde dann das Objekt dem generischen Client bereits bei der Entwicklung der Anwendung zur Verfügung steht.

Aktuell arbeitet die OPC-Foundation an neuen Spezifikationen die neue Anwendungsszenarien ermöglichen könnten. Beispielsweise schafft OPC UA über die echtzeitfähige Kommunikation via Time Sensitive Networking (TSN) neue Möglichkeiten. Dadurch kann z.B. eine Echtzeit-Kommunikation zwischen Steuerungen mittels OPC UA realisiert werden. Dann bekommen ebenfalls Themen wie Safety-Funktionen über OPC UA eine ganz neue Relevanz. Durch diese möglichen Spezifikationen könnte der relevante Anwendungsfall Safety in der Kommunikation zwischen Steuerungen und z. B. Antrieben ebenfalls durch OPC UA umgesetzt werden. OPC UA könnte dadurch ein globaler Kommunikationsstandard für die Kommunikation zwischen allen Hierarchieebenen des RAMI 4.0 werden (Abb. 2.1). Die Zukunft wird jedoch erst zeigen können, welche Anwendungsbereiche sich durch die Kommunikation über OPC UA in der Industrie etablieren werden.

Literaturverzeichnis

[ascolab 2018a] ASCOLAB: *OPC UA Anwendungen*. 2018. – URL http://www.ascolab.com/de/unified-architecture/anwendungen.html. – Zugriff: 22.03.2018

[ascolab 2018b] ASCOLAB: *OPC UA Protokolle*. 2018. – URL http://www.ascolab.com/de/unified-architecture/protokolle.html. – Zugriff: 22.03.2018

[BMWi 2013] BMWI: *Zukunftsbild Industrie 4.0*. 2013. – URL https://www.bmbf.de/pub/Zukunftsbild_Industrie_4.0.pdf. – Zugriff: 29.01.2018

[BMWi 2016] BMWI: *Interaktionsmodell für Industrie 4.0-Komponenten*. 2016. – URL https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/interaktionsmodell-I40-komponenten.pdf?__blob=publicationFile&v=16. – Zugriff: 22.01.2018

[BMWi 2017] BMWI: *Beziehungen zwischen I4.0 Komponenten Verbundkomponenten und intelligente Produktion*. 2017. – URL http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/beziehungen-%20i40-komponenten.pdf?__blob=publicationFile&v=6. – Zugriff: 05.12.2017

[BMWi 2018] BMWI: *Hintergrund zur Plattform Industrie 4.0*. 2018. – URL https://www.plattform-i40.de/I40/Navigation/DE/Plattform/Plattform-Industrie-40/plattform-industrie-40.html. – Zugriff: 29.03.2018

[Lange 2014] LANGE, Jürgen: *OPC Von Data Access bis Unified Architecture*. Vde Verlag GmbH, 2014. – ISBN 978-3-8007-3506-8

[Mahnke u. a. 2009] MAHNKE, Wolfgang ; DAMM, Matthias ; LEITNER, Stefan-Helmut: *OPC Unified Architecture*. Springer, 2009. – ISBN 978-3540688983

[OPC-Foundation 2018] OPC-FOUNDATION: *What is OPC?* 2018. – URL https://opcfoundation.org/about/what-is-opc/. – Zugriff: 07.03.2018

[Plattform Industrie 4.0 2015] PLATTFORM INDUSTRIE 4.0: *Umsetzungsstrategie Industrie 4.0*. 2015. – URL https://www.bitkom.org/noindex/Publikationen/2015/Leitfaden/Umsetzungsstrategie-Industrie-40/150410-Umsetzungsstrategie-0.pdf. – Zugriff: 05.12.2017

[Plattform Industrie 4.0 2016] PLATTFORM INDUSTRIE 4.0: *Referenzarchitekturmodell Industrie 4.0 - Eine Einführung*. April 2016. – URL http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/rami40-eine-einfuehrung.pdf?__blob=publicationFile&v=9. – Zugriff: 05.12.2017

[Schwab 2016] SCHWAB, Klaus: *Die Vierte Industrielle Revolution*. Pantheon, 2016. – ISBN 978-3-570-55345-9

[Siemens AG 2008] SIEMENS AG: *Gerätehandbuch SENTRON Multifunktionsmessgerät PAC3200*. Februar 2008. – URL https://support.industry.siemens.com/cs/ww/de/view/26504150. – Zugriff: 06.05.2018

[Siemens AG 2017] SIEMENS AG: *SIMATIC S7-1500, ET 200MP, ET 200SP, ET 200AL, ET 200pro Kommunikation*. Dezember 2017. – URL https://support.industry.siemens.com/cs/ww/de/view/59192925. – Zugriff: 07.05.2018

[Siemens AG 2018a] SIEMENS AG: *OPC UA .NET Client für den SIMATIC S7-1500 OPC UA Server*. März 2018. – URL https://support.industry.siemens.com/cs/ww/de/view/109737901. – Zugriff: 16.04.2018

[Siemens AG 2018b] SIEMENS AG: *SIMATIC Ident RFID-Systeme SIMATIC RF650R/RF680R/RF685R*. März 2018. – URL https://support.industry.siemens.com/cs/ww/de/view/109756191. – Zugriff: 06.05.2018

[VDMA und IOSB-INA 2017] VDMA ; IOSB-INA: *Industrie 4.0 Kommunikation mit OPC UA*. 2017. – URL http://industrie40.vdma.org/documents/4214230/16617345/1492669959563_2017_Leitfaden_OPC_UA_LR.pdf/f4ddb36f-72b5-43fc-953a-ca24d2f50840. – Zugriff: 05.12.2017

[Wikipedia 2018] WIKIPEDIA: 2018. – URL https://de.wikipedia.org/wiki/Automatisierungspyramide. – Zugriff: 03.04.2018

[Zehl 2018] ZEHL, Sven: *Was Industrie 4.0 (für uns) ist*. 2018. – URL https://www.bitkom.org/Themen/Digitale-Transformation-Branchen/Industrie-40/Was-ist-Industrie-40-2.html. – Zugriff: 29.03.2018

[ZVEI 2015] ZVEI: *Das Referenzarchitekturmodell Industrie 4.0*. 2015. – URL https://www.zvei.org/fileadmin/user_upload/Themen/Industrie_4.0/Das_Referenzarchitekturmodell_RAMI_4.0_und_die_Industrie_

4.0-Komponente/pdf/ZVEI-Faktenblatt-Industrie4_0-RAMI-4_0.pdf. – Zugriff: 22.01.2018

[ZVEI 2016] ZVEI: *Welche Kriterien müssen Industrie-4.0-Produkte erfüllen?* 2016. – URL https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2016/November/Welche_Kriterien_muessen_Industrie-4.0-Produkte_erfuellen_/ZVEI-LF_Welche_Kriterien_muessen_I_4.0_Produkte_erfuellen_17.03.17.pdf. – Zugriff: 05.12.2017

Anhang

A.1: Softwaredokumentation OPC UA Server

A.2: Softwaredokumentation OPC UA Client

A.3: XML-Datei der Informationsmodellierungen

A.4: TIA Portal Projekt Archiv

A.5: OPC UA Client - Visual Studio Projekt

Die Anhänge A.3, A.4 und A.5 befinden sich auf einer Daten-CD und sind bei Prof. Dr.-Ing. Holger Gräßner oder Prof. Dr.-Ing. Florian Wenck einzusehen.

A.1: Softwaredokumentation OPC UA Server

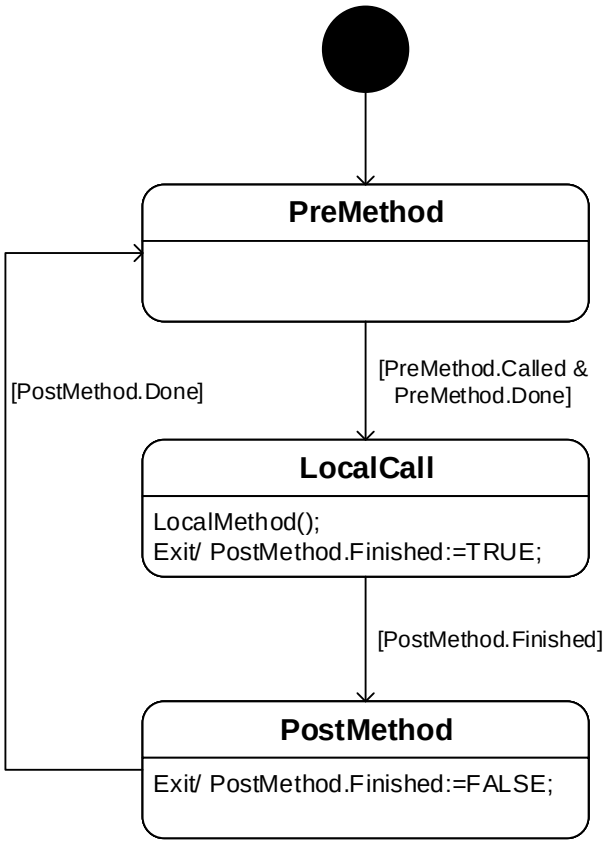

Abbildung A.1: OPC UA Methodenhandling UML State Chart

Bausteinname	Bausteintyp	Beschreibung
	Tire Production Machine	
Main_tire_production_machine	OB	Aufruf der Funktionsbausteine
Tire_production_machine_info	DB	Allgemeine Informationen zu der Maschine
	Control Program	
Activate_sleep_mode_production	FC	Aktivierung des Sleepmode vormerken
Deactivate_sleep_mode_production	FC	Deaktivierung des Sleepmode vormerken
Delivery	FC	Ablageanfrage stellen
Distance_sensor	FC	Vorverarbeitung eines Laser-Sensorwertes
Tire_production	FB	Petrinetz zur Steuerung des Prozesses
	Job Management	
Add_tire_job	FC	Anfrage eines Produktionauftrages
Control_add_transport_job	FB	Anträge für Transportaufträge stellen
	OPC UA Methods	
OPC_UA_activate_sleep_mode_production	FB	OPC UA Methode und Aufruf der lokalen Funktion
OPC_UA_add_tire_job	FB	OPC UA Methode und Aufruf der lokalen Funktion
OPC_UA_deactivate_sleep_mode_production	FB	OPC UA Methode und Aufruf der lokalen Funktion
OPC_UA_delivery	FB	OPC UA Methode und Aufruf der lokalen Funktion

Tabelle A.1: Reifenproduktionsmaschine Steuerungsprogramm Übersicht

Bausteinname	Bausteintyp	Beschreibung
	Transport	
Main_transport	OB	Aufruf der Funktionsbausteine
Transport_info_DB	DB	Allgemeine Informationen zu der Maschine
	Control Program	
Activate_sleep_mode_transport	FC	Aktivierung des Sleepmode vormerken
Deactivate_sleep_mode_transport	FC	Deaktivierung des Sleepmode vormerken
Transport_control	FB	UML-State-Chart zur Steuerung des Prozesses
	Job Management	
Add_transport_job	FC	Anfrage eines Transportauftrages
Transport_job_DB	DB	Datenspeicher für die Auftragsverwaltung
	OPC UA Methods	
OPC_UA_activate_sleep_mode_transport	FB	OPC UA Methode und Aufruf der lokalen Funktion
OPC_UA_add_transport_job	FB	OPC UA Methode und Aufruf der lokalen Funktion
OPC_UA_deactivate_sleep_mode_transport	FB	OPC UA Methode und Aufruf der lokalen Funktion

Tabelle A.2: Transportsystem Steuerungsprogramm Übersicht

Bausteinname	Bausteintyp	Beschreibung
	Warehouse	
Main_warehouse	OB	Aufruf der Funktionsbausteine
Warehouse_info_DB	DB	Allgemeine Informationen zu der Maschine
	Control Program	
Activate_sleep_mode_warehouse	FC	Aktivierung des Sleepmode vormerken
Deactivate_sleep_mode_warehouse	FC	Deaktivierung des Sleepmode vormerken
Warehouse_control	FB	UML-State-Chart zur Steuerung des Prozesses
	Job Management	
Warehouse_provide_tire_material	FC	Anfrage zum Ausschub eines Stückguts
Warehouse_tire_material_store	FC	Anfrage zum Einlagern eines neuen Stückguts
Warehouse_DB	DB	Datenspeicher für das Warenlager
	OPC UA Methods	
OPC_UA_activate_sleep_mode_warehouse	FB	OPC UA Methode und Aufruf der lokalen Funktion
OPC_UA_add_transport_job	FB	OPC UA Methode und Aufruf der lokalen Funktion
OPC_UA_deactivate_sleep_mode_transport	FB	OPC UA Methode und Aufruf der lokalen Funktion

Tabelle A.3: Warenlager Steuerungsprogramm Übersicht

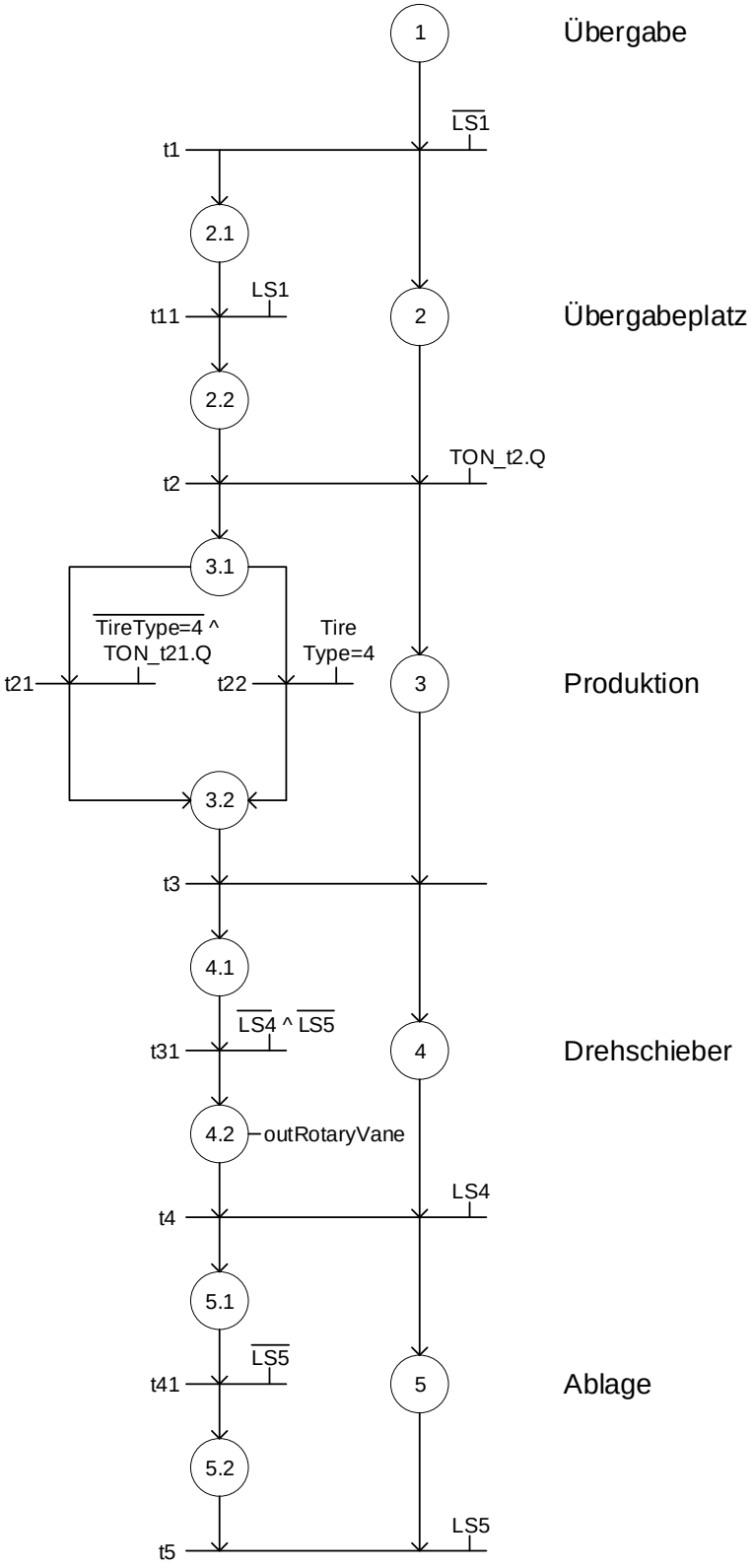

Abbildung A.2: Transportsteuerung UML State Chart

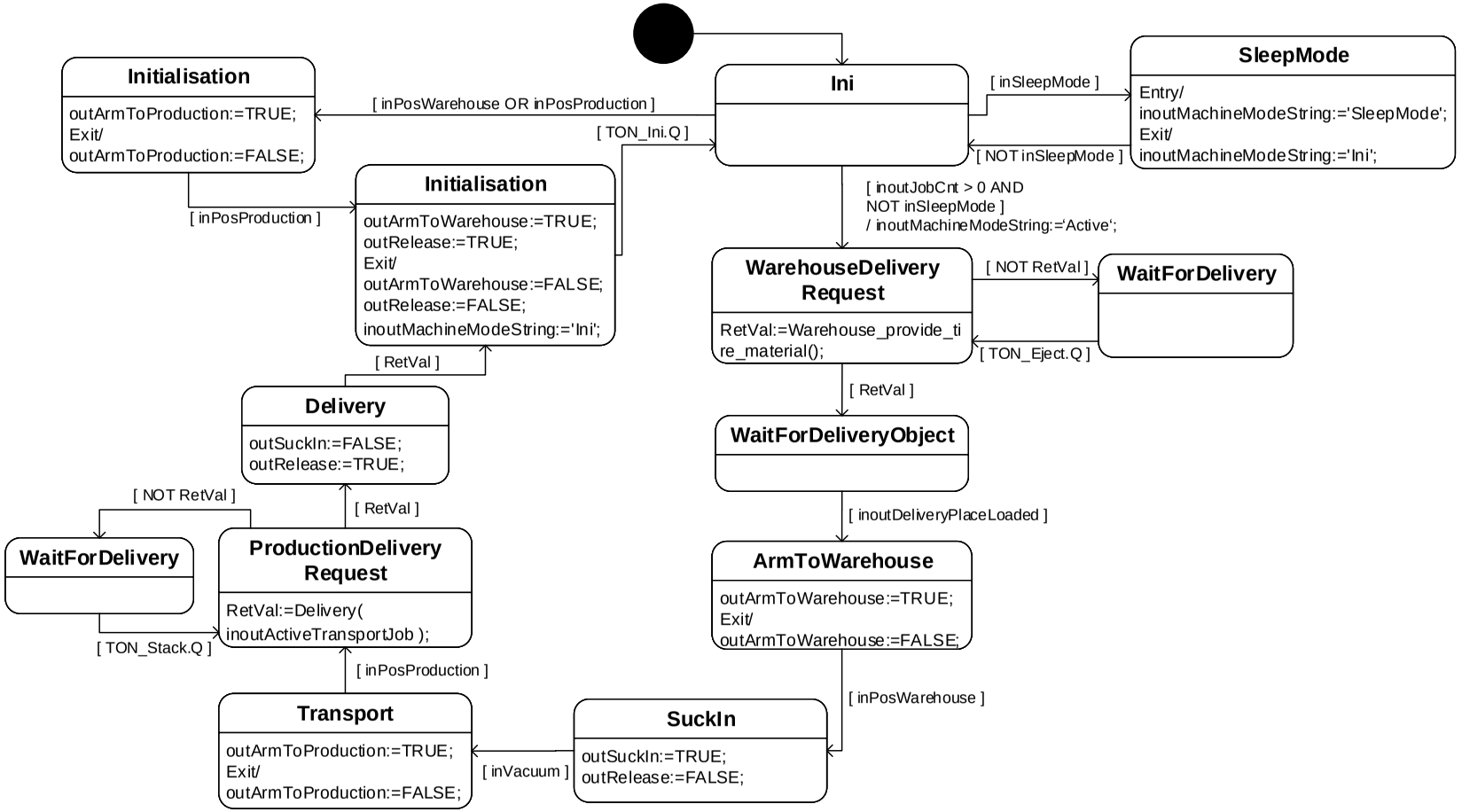

Abbildung A.3: Transportsteuerung UML State Chart

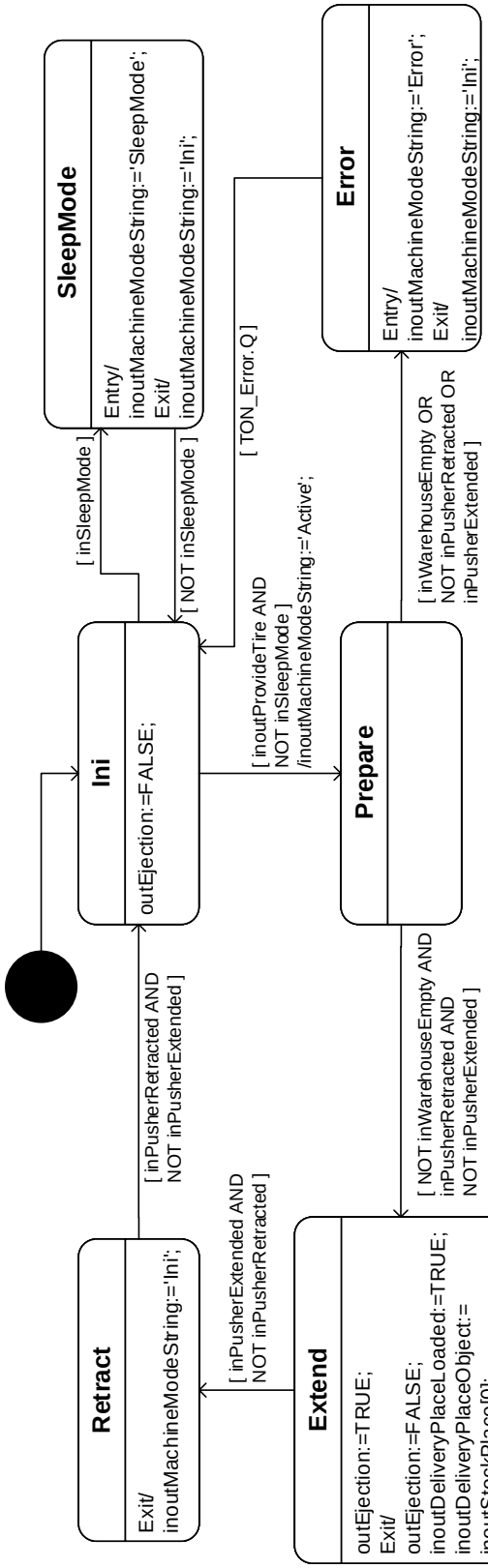

Abbildung A.4: Warenlager UML State Chart

A.2: Softwaredokumentation OPC UA Client

Abbildung A.5: Klassendiagramm UAClientHelperAPI

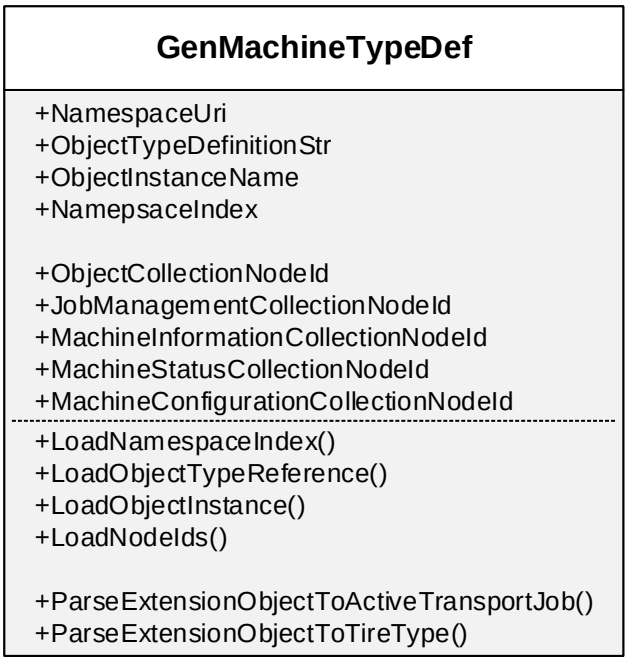

Abbildung A.6: Klassendiagramm GenMachineTypeDef

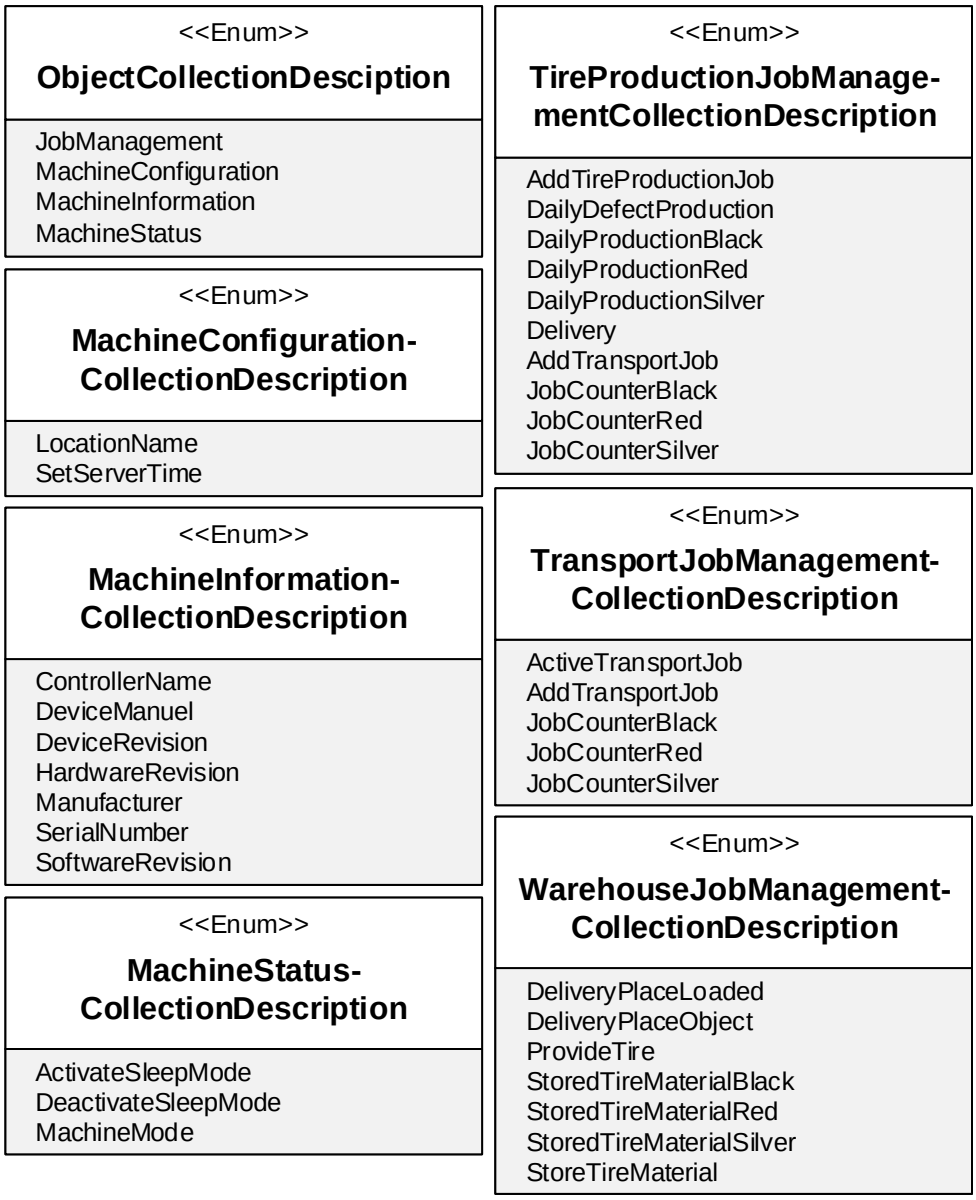

Abbildung A.7: Enums des Namespace

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 23. Mai 2018	
Ort, Datum	Unterschrift