TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Institut für
Fertigungstechnik und
Hochleistungslasertechnik

Diplomarbeit

Sensor- und Steuerungsintegration mit MTConnect

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines

Diplom-Ingenieurs unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Burkhard Kittl

und unter der Betreuung von

Univ.Ass. Dipl.-Ing. Iman Ayatollahi

(Institut für Fertigungstechnik und Hochleistungslasertechnik)

eingereicht an der Technischen Universität Wien

Fakultät für Maschinenwesen und Betriebswissenschaften

von

Thomas Trautner, BSc.

1328846

Redtenbachergasse 54/5

1160 Wien

Wien, im Juni 2016

_____

Thomas Trautner

Ich habe zur Kenntnis genommen, dass ich zur Drucklegung meiner Arbeit unter der Bezeichnung

Diplomarbeit

nur mit Bewilligung der Prüfungskommission berechtigt bin.

Ich erkläre weiters Eides statt, dass ich meine Diplomarbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen selbstständig ausgeführt habe und alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, genannt habe.

Weiters erkläre ich, dass ich dieses Diplomarbeitsthema bisher weder im In- noch Ausland (einer Beurteilerin/einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe und dass diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.

Wien, im Juni 2016

_____

Thomas Trautner

Danksagung

Zu Beginn meiner Diplomarbeit möchte ich mich herzlichst bei meinen Betreuern Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Burkhard Kittl und Univ.Ass. Dipl.-Ing. Iman Ayatollahi für ihre Anregung zu diesem Thema bedanken. Durch ihre Unterstützung und weitere Einbindung in aktuelle Projekte haben sie in mir die Faszination für das Gebiet der Fertigungstechnik und die Lösung nahender technologischer Hürden geweckt.

Außerdem gilt mein Dank meiner Familie, die mich vor und während meines Studiums finanziell und moralisch unterstützt haben und mir immer tatkräftig zur Seite standen.

Kurzfassung

Obwohl semantikbasierte Kommunikationsstandards wie MTConnect und OPC UA (OPC Unified Architecture) keine Neuheit sind und deren Entwicklung im Fall von MTConnect besonders im Bereich der Werkzeugmaschinen stark vorangetrieben wurde, ist die Anzahl der Maschinen, die eine solche Schnittstelle anbieten noch sehr überschaubar.

MTConnect bietet hierzu ein standardisiertes Schema für die Modellierung von Werkzeugmaschinen. Das dabei erstellte Modell wird von einem lokalen Webserver, dem Agent, im Netzwerk im XML (Extensible Markup Language) Format bereitgestellt und mit Prozessdaten angereichert. Diese erhält der Agent durch ein peripheres Software oder Hardwareelement, dem Adapter, welches die Daten über eine maschinenspezifische Schnittstelle ausliest, in eine dem Standard entsprechende Einheit oder Struktur umwandelt und nach einem standardisierten Protokoll an den Agent sendet. Der Agent speichert diese Daten in einem Pufferspeicher zwischen und stellt sie für Anfragen von Netzwerkclients bereit. Die Fähigkeit zur Verwendung eines mit Prozessdaten angereicherten Modells als virtuelle Repräsentation der realen Maschinenstruktur erlaubt eine einfache Möglichkeit zur Rekonfiguration und Erweiterung bestehender Systeme in diesem Modell. Vor allem ermöglicht es dadurch neben der Verwendung von standardisierten Gerätearchitekturen auch individuelle Anpassungen.

Diese Arbeit beschäftigt sich mit der Modellierung und der Integration einer EMCO Maxxturn 45 Drehmaschine in einen MTConnect Agent und der Erweiterung dieses Gerätemodells um zusätzliche Messgeräte. Der Adapter für die Drehmaschine soll von einem MTConnect Software Hersteller erworben werden und wird direkt auf der Steuerung der Maschine installiert. Für die Messgeräte wurde ein Adapter basierend auf einem freien Software Development Kit (SDK) von MTConnect programmiert. Dieser nutzt außerdem eine weiter freie Softwarebibliothek für die Kommunikation mit den Messgeräten über ModbusTCP.

Abstract

Although semantic-based communication standards such as MTConnect and OPC UA (OPC Unified Architecture) are not a novelty and their development has greatly advanced, particularly with MTConnect in the field of machine tools, the number of machines that offer such an interface is very small.

MTConnect therefore offers a standard format for the modeling of machine tools. The model thereby created will be provided by a local web server, the Agent, on the network in XML (Extensible Markup Language) format, and enriched with process data. The Agent receives this data through a peripheral software or hardware element, the Adapter, which reads data through a machine-specific interface, converts it to a unit or structure that corresponds to the standard and streams it, based on a standardized protocol, to the agent. The Agent stores the data in a buffer memory and provides it for requests from network clients. The ability to use a model enriched with process data as a virtual representation oft the real machine structure provides an easy way for reconfiguration and expansion of existing systems in this model. Above all it enables individual adjustments in addition to the use of standardized equipment architectures.

This thesis deals with the modeling and the integration of an EMCO Maxxturn 45 lathe into an MTConnect Agent and the extension of this device model to include additional measuring devices. The adapter for the lathe is to be acquired from an MTConnect software manufacturer and is installed directly on the machine's control unit. For the measuring devices, an adapter was programmed based on a free software development kit (SDK) by MTConnect. This adapter further uses another free software library for communication with the measuring devices through ModbusTCP.

Inhaltsverzeichnis

1 Einleitung

In der heutigen Industrielandschaft wird mit Vorhaben wie Internet of Things (IoT) und der davon für die Industrie spezifischen Ausprägung Industrie 4.0 (I4.0) [1] die Vernetzung aller in der Produktion notwendigen Gegenstände angestrebt. Auf Basis dieser Vernetzung sollen intelligente Systeme entstehen, die größtenteils selbstständig operieren, miteinander interagieren, sich somit eigenständig an Veränderungen anpassen können und damit eine Verbesserung der Wertschöpfung generieren. Neben einer umfassenden Integration der relevanten Informationen dieser Gegenstände in Modelle zur virtuellen Repräsentation ist vor allem die Notwendigkeit gemeinsamer Standards und Kommunikationsschnittstellen ein wichtiger Faktor dieser Entwicklungen. Dass die Vernetzung von Produktion mit Prozessen, wie der Abwicklung von kundenspezifischen Aufträgen, eine notwendige Entwicklung darstellt, zeigt die immer noch zunehmende Individualisierung von Produkten, die bis hin zur Losgröße eins eine zunehmende Komplexität in der Produktion bedeutet.

Einen Lösungsansatz für die Standardisierung der Kommunikation könnte der Standard MTConnect bieten, welcher besonders durch einfache Kommunikation und ein einheitliches Informationsmodell für die Steuerungsdaten einer Werkzeugmaschine versucht, die Komplexität bei der Modellierung von Gerätedatenstrukturen und der Integration des Kommunikationsstandards zu senken. Außerdem können die Kosten durch Nutzung freier Software und oft schon bestehender TCP/IP (Transmission Control Protocol / Internet Protocol) Netzwerkstrukturen stark reduziert werden. Diese Arbeit behandelt daher die Integration des MTConnect Standards an einer Drehmaschine mit zusätzlicher Erweiterung um externe Sensoren zur Erfassung von elektrischen Leistungen, um die Maschinendaten im Netzwerk verfügbar zu machen und so eine einfache Grundlage zur Informationsverarbeitung zu bieten.

1.1 Stand der Technik

Aktuelle Bestrebungen wie RAMI4.0 (Referenzarchitekturmodell Industrie 4.0) versuchen ein Framework für Entwicklungen im Rahmen von I4.0 zu schaffen. Dabei entsteht ein Modell, das die vertikale Integration bis hin zum Werkstück und die horizontale Integration entlang der Wertschöpfungskette bzw. über verschiedene Standorte oder Wertschöpfungsnetzwerke einbezieht. Durchgängiges Engineering soll jederzeit den Zugriff auf anfallende Daten über das Netzwerk erlauben [1].

Kern dieser Entwicklungen wird dabei u. a. die Integration vorhandener Standards in RAMI4.0 sein. Zur Realisierung der Kommunikationsschicht wurde dabei besonders der semantikbasierte Kommunikationsstandard OPC UA (OPC Unified Architecture) betrachtet. Dieser zeichnet sich vor allem durch das umfassende und erweiterbare Meta Modell aus, das eine objektorientierte Informationsmodellierung ermöglicht, mit welcher Geräte und deren Komponenten und Funktionen dynamisch modelliert werden können. OPC UA besitzt dabei die Fähigkeit diese strukturierten Informationen für Kommunikationspartner bereitzustellen. Durch die Modellierung nach definierten Typen versteht der Client nicht nur die Struktur, sondern auch die Bedeutung der einzelnen Funktionen und Datenpunkte innerhalb des Modells. Dies senkt vor allem den Konfigurationsaufwand und führt letztendlich in einem optimalen Szenario zur automatischen Erkennung und Interaktion mit einem neuen Gerät.

Als amerikanische Alternative zu OPC UA wird oft der Standard MTConnect gehandelt, wenngleich dieser durch seine Spezialisierung auf Werkzeugmaschinen nur auf eine Teilmenge der mit OPC UA modellierbaren Geräte abzielt. Genau darauf ist auch die MTConnect Geräteschemadefinition ausgelegt, die Regeln für Struktur und Datentypen von Werkzeugmaschinen festlegt, aber auch erweitert werden kann. Diese erlaubt es, ähnlich wie bei OPC UA, eine virtuelle Repräsentation eines Geräts zu modellieren, welche von einem Kommunikationspartner ausgelesen werden kann. Die Kommunikation mit den Clients übernimmt dabei ein sogenannter MTConnect Agent, der das erstellte Informationsmodell enthält und alle Clients mit den spezifisch angefragten Informationen versorgt. Die Daten hierfür erhält der Agent von einem oder mehreren MTConnect Adaptern, die die Daten über eine gerätespezifische Schnittstelle auslesen und per Socketstream an verbundene Agents weiterleiten.

Der Hersteller DMG Mori Seiki hat MTConnect bereits in seinen Werkzeugmaschinen implementiert. Dazu wird die vorhandene HMI (Human Machine Interface) Schnittstelle verwendet, um so auch ältere Geräte mit MTConnect auszustatten [2]. MTConnect muss daher nicht zwangsläufig tief in die Maschine implementiert werden, sondern kann auch auf vorhandenen Schnittstellen aufsetzen.

MTConnect ermöglicht die Umsetzung vielfältiger Konzepte und Ansätze. Denn zum einen liefert MTConnect die Möglichkeit zur Erweiterung bestehender Systeme um zusätzliche Sensoren, um eine Korrelation und Auswertung der Daten unterschiedlicher Geräte zu begünstigen. Ein entsprechendes Konzept zum Energiemonitoring wurde bereits von Vijayaraghavan und Dornfeld [3] vorgestellt, in dem simulierte Energieverbrauchsdaten mit Maschinenoperationen korreliert werden. MTConnect wird dabei als Basis für derartige Systeme diskutiert. Lynn, Chen, Locks, Nath und Kurfess [4] beschreiben die Erweiterung einer Maschine um einen Beschleunigungssensor, um den Zustand der Spindel zu diagnostizieren und über MTConnect bereitzustellen.

Andere Anwendungen beschreiben die Nutzung von MTConnect zur Entwicklung einer *Smart Machine*, um auf Basis der Datenauswertung einen optimalen Produktionsprozess zu erhalten, um Produktivität und Qualität zu steigern und Ausfälle zu senken [5]. Ein ähnlicher Ansatz von Shin, Woo und Rachuri [6] beschreibt ein *Smart Manufacturing System*, das *Machine Learning* Ansätze verwendet, um Aussagen über Energieverbräuche beim Schneiden mit verschiedenen Eingangsparametern, wie Material, Schnitttiefe und Vorschubgeschwindigkeit, zu treffen. Filleti, Silva, Silva und Ometto [7] nutzen MTConnect im Zusammenhang mit Life Cycle Assessment (LCA) um Hotspots bei der Produktion aufzudecken und darauf aufbauend bessere Produktionsstrategien zu entwickeln.

1.2 Aufgabe, Problemstellung und Vorgehensweise

Abbildung 1 zeigt die Ausgangssituation des Energiemonitorings in der Pilotfabrik der Technischen Universität Wien. Hierzu wird die SCADA (Supervisory Control and Data Aquisition) Software WinCC Open Architecture (WinCC OA) von Siemens verwendet, um die Energiedaten und einzelne Steuerungsparameter eines Drehzentrums zu visualisieren. Die elektrischen Messdaten werden von vier SentronPAC Messgeräten der Firma Siemens über das Protokoll ModbusTCP geliefert. Die Messgeräte messen einmal die Gesamtwirkleistung an der Hauptzuleitung des Gerätes und die Wirkleistungen von Antriebs-, Kühlschmierstoff- und Hydrauliksystem.

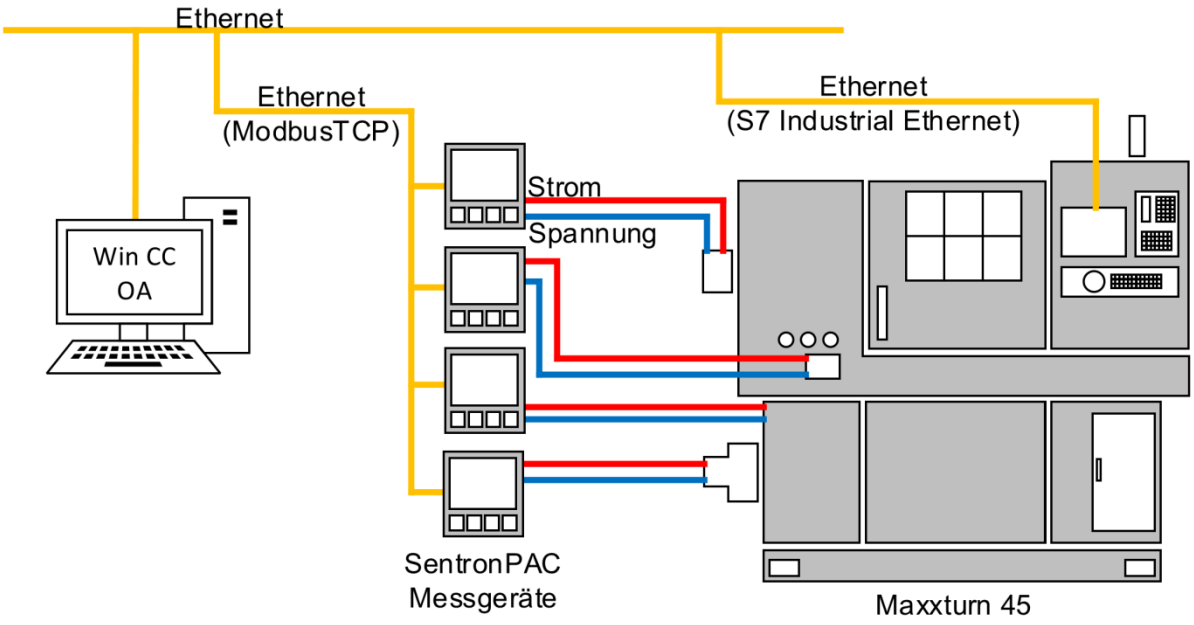

Abbildung 1: Ausgangssituation

Ziel der Arbeit ist die Integration der Mess- und Steuerungsdaten dieser Anlage in einen MTConnect Agent. Über den Agent sollen die Steuerungsdaten des Drehzentrums erfasst und bereitgestellt werden. Dazu wird eine Modellierung der Maschine nach dem Informationsmodell von MTConnect benötigt und außerdem die Anbindung der Maschine an den Agent durch einen Adapter. Diese Anbindung ist jedoch problematisch, da die Steuerung der Drehmaschine keine Schnittstelle bietet, die eine für den Standard umfassende Modellierung von Datenpunkten gewährleistet. Um die Anpassungsfähigkeit des Standards zu zeigen, soll das Modell der Maschine außerdem um zusätzliche Komponenten erweitert werden. Dazu sollen dem Agent Daten externer Messgeräte über eine Modbus TCP Schnittstelle bereitgestellt werden. Letztendlich soll so gezeigt werden, wie Daten auch von unterschiedlichen Schnittstellen in einem Agent gemeinsam modelliert und basierend auf diesem Informationsmodell standardisiert verwertet werden können.

Hierfür werden zuerst die notwendigen Grundlagen für das Verständnis des MTConnect Standards erläutert. Dazu wird zu Beginn der Standard XML (Extensible Markup Language) erläutert, der als Format für die modellbasierte Repräsentation der Maschinenstruktur von Werkzeugmaschinen nach der Syntax von MTConnect dient. Daraufhin folgt eine Beschreibung von MTConnect mit dem standardisierten Informations- und dem Kommunikationsmodell.

Im Praxisteil folgt nach einer Beschreibung des Aufbaus und Zielsystems (Abbildung 2) die Festlegung der Ziele für die Integration der Messgeräte. Als Hauptteil der Arbeit erfolgt die klassenbasierte Programmierung eines skalierbaren Adapters für die ModbusTCP Messgeräte und die Modellierung der Drehmaschine zusammen mit den Messgeräten nach dem MTConnect Standard.

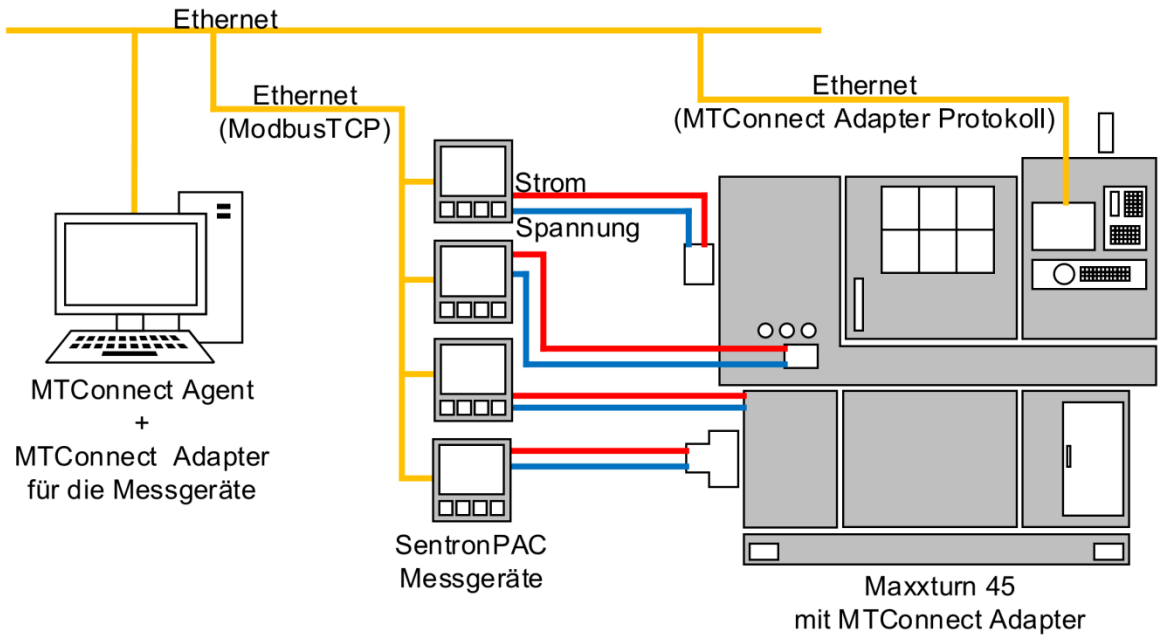

Abbildung 2: Zielsystem mit MTConnect

2 Grundlagen

In diesem Kapitel werden die grundlegenden Konzepte von MTConnect und XML (Extensible Markup Language) behandelt, die zum Verständnis und zur Anwendung des MTConnect Standards benötigt werden. In Kapitel 2.1 wird zuerst der XML Standard vorgestellt, auf dem das Informationsmodell von MTConnect basiert. MTConnect selbst wird anschließend in Kapitel 2.2 genauer erläutert.

2.1 Extensible Markup Language (XML)

Die Extensible Markup Language, im Deutschen auch erweiterbare Auszeichnungssprache genannt, wurde erstmals 1998 vom World Wide Web Consortium (W3C) vorgeschlagen. Mit XML werden Begriffen beschreibende Eigenschaften (Metadaten) hinzugefügt. Dazu können Regeln für die Sprache festgelegt und diese dann in standardisierter Form verwendet werden [8]. Durch XML werden diesen Begriffen also beschreibende und strukturierende Metadaten hinterlegt, mit denen Wörter einem bestimmten Typ zugeordnet und Zusammenhänge zwischen den Wörtern festgelegt werden können. Durch den einfachen und standardisierten Aufbau soll XML von Maschinen und von Menschen leicht lesbar und erweiterbar sein [9].

Die in Code-Snippet 1 beschriebene Liste enthält eine Reihe von Wörtern ohne die dazugehörigen Metadaten. Aus dieser Liste ist für den Menschen aufgrund seiner Sprache erkenntlich, dass es sich um Vornamen und Nachnamen handelt. Eine entsprechende Software könnte mit den Daten nur etwas anfangen, wenn die Position der Wörter fest vorgegeben ist und sie diese Daten so anhand ihrer Struktur einem programmierten Typ zuordnen kann. Würden jedoch Vor- und Nachnamen vertauscht werden oder ein Zweitname hinzukommen, so würde die Datenverarbeitung viel komplexer werden und softwaretechnisch würden ohne weitere Informationen keine sicheren Rückschlüsse auf die Art der einzelnen Wörter gezogen werden können.

```
1. Franz Kafka
2. Bertolt Brecht
3. Alfred Joseph Hitchcock
```

Code-Snippet 1: Beispiel in Textform

Wenn die in Code-Snippet 1 gezeigte Liste jedoch ins XML Format übersetzt wird (Code-Snippet 2), dann könnten die für die Software zum Verständnis notwendigen Metadaten zugeordnet werden. Die Datenverarbeitung erkennt nicht nur, dass es sich um eine Liste von Personen handelt, sondern auch welche die einzelnen

Personen sind und welche Attribute, wie Nachnamen und Vornamen, diese besitzen. Die vorher als einfache Liste dargestellten Daten sind nun durch ihre Metadaten strukturiert. Die Namen selbst werden dabei weiterhin als reiner Text interpretiert und werden als Character Data (CDATA) bezeichnet [10].

```
1.  <Personen>
2.    <Person>
3.      <Vorname Vornamennummer="1">Franz</Vorname>
4.      <Nachname>Kafka</Nachname>
5.    </Person>
6.    <Person>
7.      <Vorname Vornamennummer="1">Bertolt</Vorname>
8.      <Nachname>Brecht</Nachname>
9.    </Person>
10.   <Person>
11.     <Vorname Vornamennummer="1">Alfred</Vorname>
12.     <Vorname Vornamennummer="2">Joseph</Vorname>
13.     <Nachname>Hitchcock</Nachname>
14.   </Person>
15. </Personen>
```

Code-Snippet 2: Beispiel mit XML Metadaten

In Code-Snippet 2 wird jedoch auch der Nachteil einer solchen Auszeichnungstechnik deutlich, denn der Umfang an notwendigen Daten hat sich um ein Vielfaches erhöht. Dieser Nachteil ist schon zu Beginn der Entwicklung von XML deutlich geworden, wurde jedoch aufgrund der Möglichkeiten, die XML bietet, so hingenommen [9].

2.1.1 Aufbau von XML Dokumenten

Der Aufbau von XML Dokumenten erfolgt nach einer Hierarchie, in der einzelne XML Elemente mehrere Unterelemente, aber nur ein übergeordnetes Element besitzen können. Somit steht immer ein einzelnes Element an der Spitze dieser Struktur. Elemente werden außerdem immer von sogenannten Tags eingeschlossen [8]. Diese Start- und End-Tags bestehen aus dem Elementnamen in spitzen Klammern, wobei der End-Tag durch einen Schrägstrich vor dem Elementnamen bzw. vor der schließenden Klammer symbolisiert wird.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <!--Kommentar -->
3.  <Element Attribut="Text">
4.    <Element2 Attribut2="Text">Text</Element2>
5.    <Element3 Attribut3="Text" Attribut4="Text"/>
6.  </Element>
```

Code-Snippet 3: XML Aufbau

Im Grunde genommen gibt es in XML zwei Arten von Blöcken: Elemente und Attribute. Elemente können aus mehreren weiteren Elementen bestehen, können mehrere Attribute oder einfach nur reinen Text besitzen. Die Elemente haben die

Aufgabe Objekte oder eine Hierarchie von Objekten abzubilden und diese so zu strukturieren. Attribute sind immer Teil eines Elements und enthalten mit ihrem Namen die Art der Information bzw. Eigenschaft und in Textform den Wert, also die spezifische Ausprägung dieser Information. Ein weiterer Teil von XML Dokumenten, der jedoch nicht zwingend vorgeschrieben wird, ist die XML Deklaration. Wenn sie angegeben wird, muss sie in der ersten Zeile stehen und wird durch ein Fragezeichen am Anfang und am Ende, umschlossen von spitzen Klammern, gekennzeichnet. Neben Informationen zur Versionsnummer oder der Kodierung kann sie noch weitere Verarbeitungsinformationen für einen XML Parser enthalten. Ein solcher übersetzt und verarbeitet die Informationen aus dem XML Dokument und macht sie für ein Programm bzw. eine Anwendung verfügbar [10].

2.1.2 XML Schema Definition (XSD)

Wenn XML zur Kommunikation von Maschinen verwendet werden soll, dann reicht es bei einer größeren Vielfalt der kommunizierten Parameter oft nicht aus, dass die XML Daten ohne eine genaue Richtlinie mit Metadaten ausgezeichnet werden. Es muss sichergestellt werden, dass die verwendeten Metadaten und die zugrunde liegenden Hierarchien und Definitionen der Daten bei allen Empfängern und Sendern gleich sind. Hierfür ist die Definition einer gemeinsamen Sprache notwendig, welche in Form eines XML Schemas festgelegt werden kann. Ein XML Schema kann von Maschinen gelesen und verarbeitet werden, um dem Schema entsprechende XML Dokumente zu erstellen. Zuletzt können erstellte XML Dokumente auch anhand der Schemadatei auf ihre Korrektheit entsprechend dem Schema geprüft werden. Dieser Vorgang nennt sich Validierung.

Der Aufbau einer Schema Datei entspricht einem normalen XML Dokument. Hier werden ebenso Hierarchien mittels Elementen und Attributen abgebildet [10]. In einer XSD Datei werden jedoch nur die einzelnen Elemente und Attribute, die in der XML Datei verwendet werden dürfen, beschrieben und miteinander verknüpft. Zur Beschreibung der Elemente gehört in jedem Fall der Elementname und, falls vorhanden, die möglichen Unterelemente und Attribute. Dabei wird bei jeder neuen Elementtypendefinition zwischen einem *complexType*- und einem *simpleType*-Element unterschieden. Bei einem complexType-Element enthält das zu beschreibende Element weitere Subelemente oder Attribute, bei einem simpleType nicht. Attribute werden daher generell als simpleType beschrieben. Durch Verketten der zugelassenen Elemente kommt die Hierarchie der Objekte zum Vorschein.

Basierend auf dem Beispiel aus Code-Snippet 2 ergibt sich die Hierarchie, die in Abbildung 3 zu sehen ist. Dabei kann das Element *Personen* aus mehreren Elementen des Typs *Person* bestehen. Eine *Person* besitzt dabei immer genau einen

Nachnamen und kann mehrere *Vornamen* besitzen, deren Position durch das Attribut Vornamennummer spezifiziert ist.

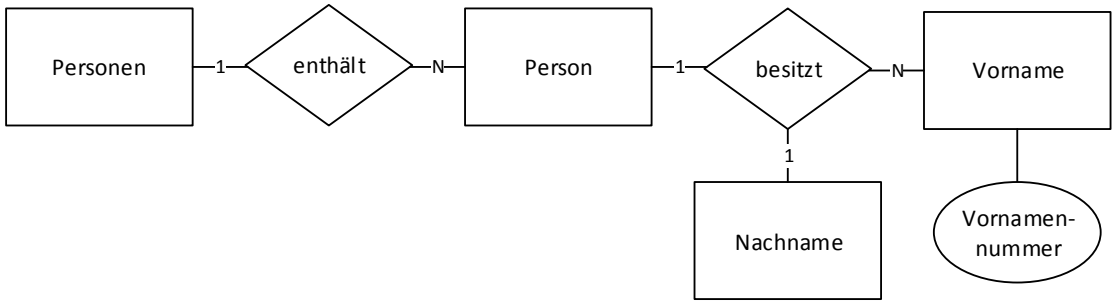

Abbildung 3: Entity Relationship Modell

Aus dem Modell in Abbildung 3 wird direkt ersichtlich, dass *Nachname* und *Vornamennummer* keine Unterelemente oder Attribute besitzen und somit dem simpleType entsprechen. Die restlichen Elemente sind stellen einen complexType dar.

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.
4.    <xs:element name="Personen" type="PersonenType"/>
5.
6.    <xs:complexType name="PersonenType">
7.      <xs:sequence>
8.        <xs:element name="Person" type="PersonType"/>
9.      </xs:sequence>
10.   </xs:complexType>
11.
12.   <xs:complexType name="PersonType">
13.     <xs:sequence>
14.       <xs:element name="Vorname" type="VornameType"/>
15.       <xs:element name="Nachname" type="NachnameType"/>
16.     </xs:sequence>
17.   </xs:complexType>
18.
19.   <xs:simpleType name="NachnameType">
20.     <xs:restriction base="xs:string"/>
21.   </xs:simpleType>
22.
23.   <xs:complexType name="VornameType">
24.     <xs:sequence>
25.       <xs:extension base="xs:string">
26.         <xs:attribute name="Vornamennummer" type="VornamennummerType"/>
27.       </xs:extension>
28.     </xs:sequence>
29.   </xs:complexType>
30.
31.   <xs:simpleType name="VornamennummerType">
32.     <xs:restriction base="xs:integer">
33.       <xs:minInclusive value="1"/>
34.       <xs:maxInclusive value="12"/>
35.     </xs:restriction>
36.   </xs:simpleType>
37. </xs:schema>
```

Code-Snippet 4: XML Schema

Das in Code-Snippet 4 gezeigte Schema enthält die im vorherigen Modell beschriebenen vier Elemente in den Zeilen 4, 8, 14 und 15. Diese sind enthalten jeweils die Attribute *name* und *type*. Das Attribut *type* ist hierbei wichtig, da es auf ein complexType- oder simpleType-Element referenziert. Damit kann wiederrum die Struktur von Unterelementen und -attributen beschrieben und mit weiteren Type-Elementen verknüpft werden. Zusätzlich können in einem Type-Element Restriktionen, wie z. B. die erlaubten Datentypen, für den Wert eines Elements bzw. Attributs festgelegt werden. Hierfür gibt es in XML bereits eine Reihe an built-in Datentypen, wie *DateTime*, *Integer*, *String* oder *Double* [11]. Restriktionen können dabei ebenfalls die erlaubten Werte auf einen bestimmten Bereich oder eine bestimmte Zeichenkombination begrenzen (Zeile 33 und 34).

2.1.3 Namespaces

Mittels eines Namensraumes (*Namespace*) können Elemente in XML Dokumenten unter einem bestimmten Block gruppiert werden. Bei der Verwendung von Elementnamen aus verschiedenen Namensräumen können diese Elemente dann einem bestimmten Raum zugeordnet werden. Das ist besonders dann notwendig, wenn Elemente aus verschiedenen XML Strukturen stammen und die gleichen Namen, aber unterschiedliche Bedeutungen besitzen [9]. In einem XML Dokument wird dazu ein Namensraum mit einem bestimmten Bezeichner verbunden, welcher direkt vor jedem zu diesem Namensraum gehörenden Element angeschrieben und mit einem Doppelpunkt getrennt wird [8].

Dies erlaubt es bereits entwickelte XML Strukturen weiterzuverwenden, ohne eine komplett neue Struktur erschaffen zu müssen. Um solche Strukturen voneinander abzugrenzen werden diese durch einen *Uniform Resource Identifier* (URI) gekennzeichnet. Mit diesem soll einer Struktur ein einzigartiger Code für dessen Identifikation zugewiesen werden. URI kann daher ein einzigartiger Name für dieses Schema sein (*Unique Resource Name* - URN) oder ein eindeutiger Ort, an dem sich diese Struktur oder Informationen zu dieser befinden, wie ein Link zu einer Website (*Uniform Resource Locator* - URL). Trotzdem muss sich unter dieser Adresse nicht zwangsläufig etwas befinden, da es sich auch nur um einen eindeutigen Namen für den Namensraum handeln kann [10].

In Code-Snippet 4 wird in Zeile 2 mit einer URL auf den Namensraum verwiesen. Der Namensraum wird mit dem Attribut *xmlns* (XML Namespace) eingeleitet und erhält den Bezeichner *xs*. Mit diesem Präfix werden nun alle Elemente referenziert, die aus dem Schema mit diesem Namensraum entnommen werden. Ebenso kann der Wert eines Attributes, wie der Datentyp String in Zeile 20, mit einem Namensraum verknüpft werden.

2.2 MTConnect

MTConnect ist ein unidirektionaler Kommunikationsstandard, der speziell im Bereich der Werkzeugmaschinen eingesetzt wird, um Daten von Maschinen im Netzwerk darzustellen. MTConnect setzt dabei für die Darstellung und Strukturierung von Daten auf den XML Standard und für die Kommunikation selbst auf den Internetstandard Hypertext Transfer Protocol (HTTP). Außerdem bietet MTConnect standardisierte Modelle für Gerätedatenstrukturen, Kommunikation und Assets als XSD Dateien an, welche die Konfiguration und Vereinheitlichung von Daten verschiedenster Geräte erheblich vereinfachen. Durch die Möglichkeit diese Schemata vollständig anzupassen wird es dem Anwender ermöglicht, diese um Begriffe zu erweitern, um u. a. Datentypen, Komponenten oder neue Assets hinzuzufügen [12]. Entwickelt wurde der Standard basierend auf Untersuchungen von bereits vorhandenen Standards, wie Cam-X (Computer Aided Manufacturing using XML) und OPC [5].

Eine weitere Besonderheit von MTConnect ist, dass der Standard eine gebührenfreie Nutzung verspricht. Es entfallen also keine Lizenzgebühren für den Standard selbst. Kosten können also nur für Dienste, wie Implementation oder Software Engineering, durch einen Dritten anfallen [12]. Jedoch bietet MTConnect umfangreiche Softwarebibliotheken und Programme mit freiem Zugang und freier Verfügung an, mit denen die Entwicklung von Software vereinfacht und relativ günstig wird.

In diesem Kapitel wird MTConnect entsprechend dem Standard behandelt, wie dieser in der aktuellen Version 1.3.1 (Juni 2015) vom MTConnect Institute vorgegeben wird. Der Standard besteht aus vier Teilen, von denen Teil 1 eine Übersicht über den Standard bietet, Teil 2 den Aufbau von Gerätedaten im Agent beschreibt und Teil 3 die Kommunikation zwischen den Netzwerkelementen behandelt. Teil 4 beschreibt die Verwendung von Assets, also materiellen oder immateriellen Objekten, die zwar etwas mit dem Fertigungsprozess zu tun haben, aber einer Maschine nicht als eine feste Komponente zugeordnet werden können, wie Schnittwerkzeuge oder Werkstücke [13] [14].

2.2.1 Netzwerkkomponenten

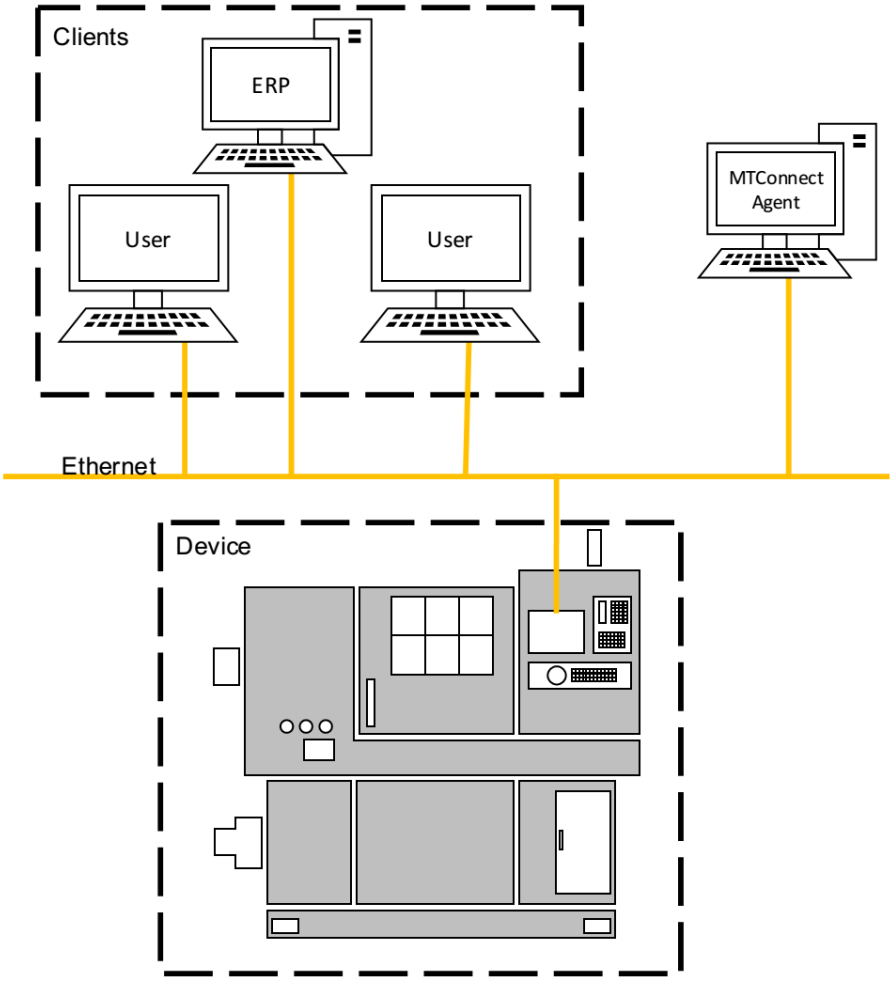

Abbildung 4: MTConnect Netzwerkkomponenten

Die grundlegenden Komponenten, die für den Einsatz von MTConnect notwendig sind, sind ein Netzwerk, ein Gerät, dessen Daten bereitgestellt werden sollen (Device), ein MTConnect Agent, der diese Daten im Netzwerk bereitstellt und ein Client, der die Daten von diesem Agent anfordern kann und diese dem Anwender entsprechend darstellt (Abbildung 4).

2.2.1.1 MTConnect Agent

Der Agent hat die Aufgabe die Daten, die ihm über das Netzwerk von einem oder mehreren Geräten gesendet werden, zwischenzuspeichern, aufzubereiten und in standardisierter Form für Anfragen (Requests) bereitzustellen. Ein MTConnect Agent muss dabei mindestens ein Gerät repräsentieren.

Ein wichtiger Punkt für die Verwendung eines Agents ist die Entscheidung über die zugewiesene Speicherkapazität. Da ein Agent nur eine endliche Anzahl an Daten speichern und zur Verfügung stellen kann, schreibt der Standard einen Pufferspeicher für den Agent vor. Bei Erreichen der maximalen Kapazität wird für

jedes neue Datenelement das älteste Datenelement entfernt. Ausschlaggebend hierfür ist die Sequenznummer, mit der jedem neuen Datenelement ein fortlaufender Primärschlüssel zugewiesen wird [15]. Die Dauer der Verfügbarkeit von Daten ist also von der Puffergröße abhängig.

Im Grunde genommen ist es aus Anwendersicht nicht das Ziel möglichst viele Daten im Agent zu sammeln. Der Agent soll nur eine vereinheitlichende Schnittstelle für den Anwender bzw. die Software darstellen. Durch das beschreibende Informationsmodell von MTConnect können Anwendungen die Struktur von Daten interpretieren und diese so standardisiert weiterverwenden, um entsprechende Statistiken zu erstellen oder spezifische Datenpunkte in Datenbanken zu übertragen.

2.2.1.2 Client

Ein Client ist in der Regel eine Anwendung auf einem Rechner im Netzwerk, die per HTTP auf spezifische Daten des Agents zugreift und diese weiter verarbeitet. Es kann aber auch eine Person sein, die über einen Internetbrowser auf den Agent zugreift und den aktuellen Status der Maschinen abfragt.

2.2.1.3 Netzwerk

Das Netzwerk stellt die Verbindung zwischen den einzelnen Elementen her. Da die Kommunikation zwischen Agent und Client auf dem HTTP Protokoll basiert, können für die Kommunikation herkömmliche TCP/IP Kommunikationsverfahren verwendet werden (Abbildung 5). Gleiches gilt für die Kommunikation zwischen Agent und Device, wenngleich hier ein eigenes vereinfachtes Protokoll zum Einsatz kommt.

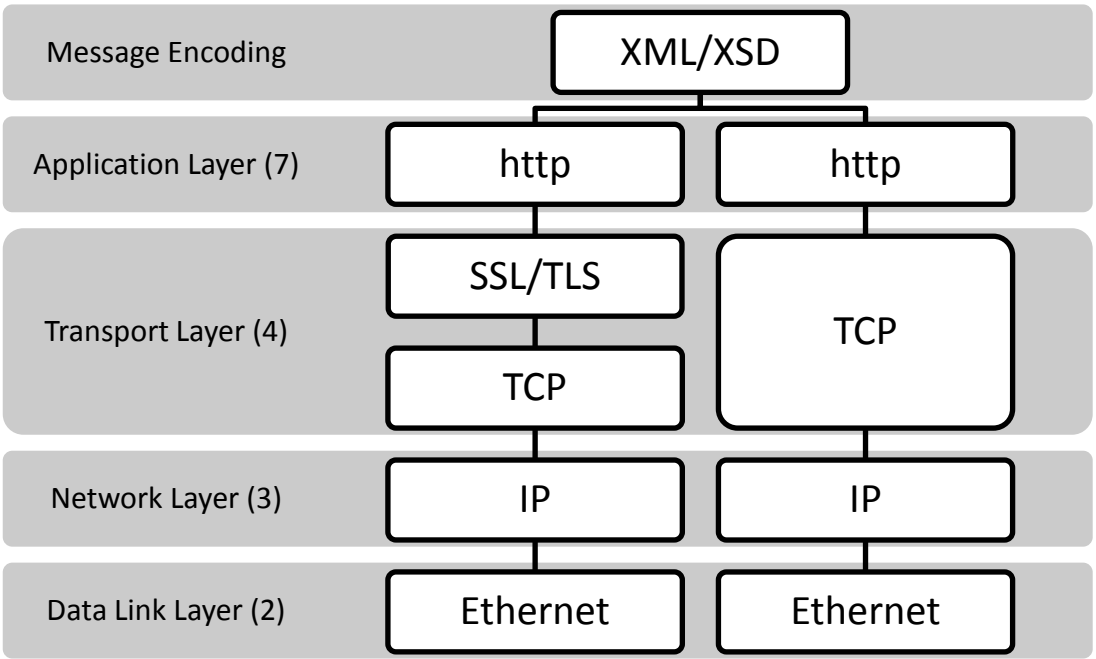

Abbildung 5: MTConnect im Open Systems Interconnection (OSI) Referenzmodell nach [16]

2.2.1.4 Device

Das Gerät selbst ist der eigentliche Lieferant der Daten, die mit MTConnect bereitgestellt werden. Dabei muss die Datenstruktur des Gerätes dem jeweiligen Agent im XML Format bereitgestellt werden, damit dieser die Daten dem Informationsmodell zuordnen kann.

Es kann sich bei dem Gerät um jede Art von Maschine oder Gegenstand handeln, der Daten erzeugen und diese über eine Schnittstelle bereitstellen kann. Der Aufbau in Abbildung 4 setzt voraus, dass das Gerät bzw. die Maschine die Kommunikation mit dem Agent bereits nativ beherrscht, was jedoch zum aktuellen Zeitpunkt bei kaum einem Gerät der Fall ist. In diesem Fall ist ein zusätzliches Objekt als Schnittstelle zwischen Gerät und Agent notwendig, welches die Kommunikation durchführt. Diese Schnittstelle heißt MTConnect Adapter und ist in Abbildung 6 dargestellt.

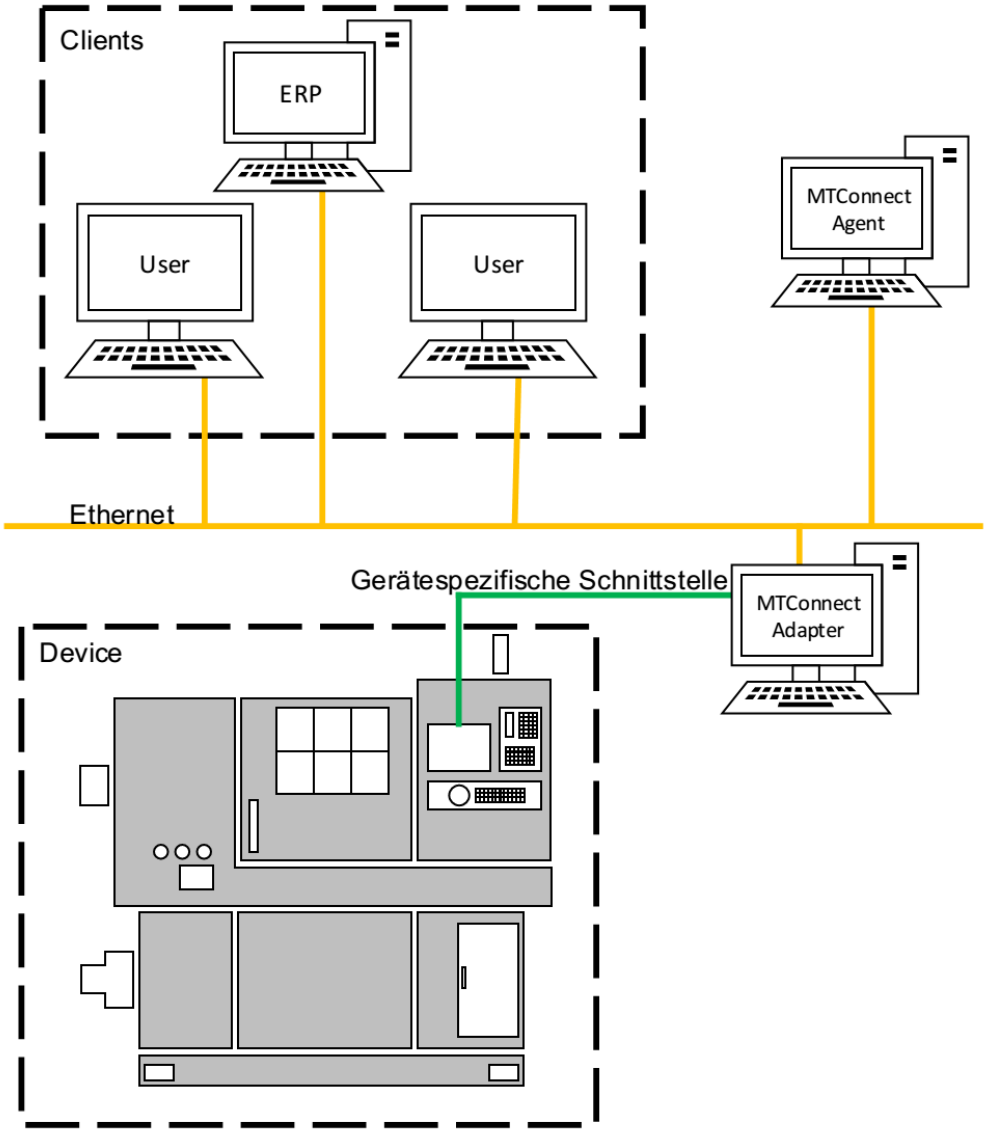

Abbildung 6: MTConnect Netzwerkkomponenten mit Adapter

2.2.1.5 MTConnect Adapter

Ein MTConnect Adapter ist eine Komponente, die die Kommunikation für das Gerät mit dem Agent übernimmt. Dabei wird der Adapter über eine gerätespezifische Schnittstelle mit dem Gerät verbunden. Der Adapter erhält dadurch die Daten aus dem Gerät und verarbeitet diese so, dass sie dem MTConnect Adapter Protokoll *Simple Hierarchical Data Representation* (SHDR) entsprechen. Dieses Protokoll enthält für einzelne Datenpunkte nur einen Zeitstempel, den Identifier des Datenpunktes und einen bzw. eine Reihe von zugehörigen Werten, getrennt mit einem senkrechten Strich (Verkettungszeichen). Über eine Socketkommunikation streamt der Adapter jedem verbundenen Agent diese Daten. Der Adapter kann dabei ein zusätzliches Hardwareelement für die Maschine oder nur softwareseitig in der Maschine selbst implementiert sein.

Prinzipiell ist es nicht notwendig, dass mehrere Rechner im Netzwerk vorhanden sind. Es ist ebenso möglich, dass Agent, Adapter und Client auf demselben Rechner laufen oder alle direkt in der Maschine implementiert sind.

2.2.2 Informationsmodell

Teil 2 des MTConnect Standards beschreibt das Informationsmodell und behandelt damit den datentechnischen Aufbau und die Struktur von Geräten. Dieses Modell wird vom Agent benötigt, damit er einzelne Werte, die er bei der Kommunikation erhält, den zugehörigen Datenpunkten in den Geräten zuordnen kann. Die Struktur der Daten wird im XML Format bereitgestellt und muss bei der Implementierung eines neuen Adapters/Devices im Agent hinterlegt werden. Mit dieser Struktur wird also der reale Aufbau einer Maschine in ein XML Modell überführt, welches den Clients zur Verfügung gestellt wird, damit diese die Maschinenstruktur in Informationsverarbeitungsprozesse miteinbeziehen können (Abbildung 7). Das Informationsmodell basiert auf einem XML Schema (MTConnectDevices)[1], welches den erlaubten Aufbau von Geräten beschreibt und, falls notwendig, um entsprechende Komponenten erweitert werden kann.

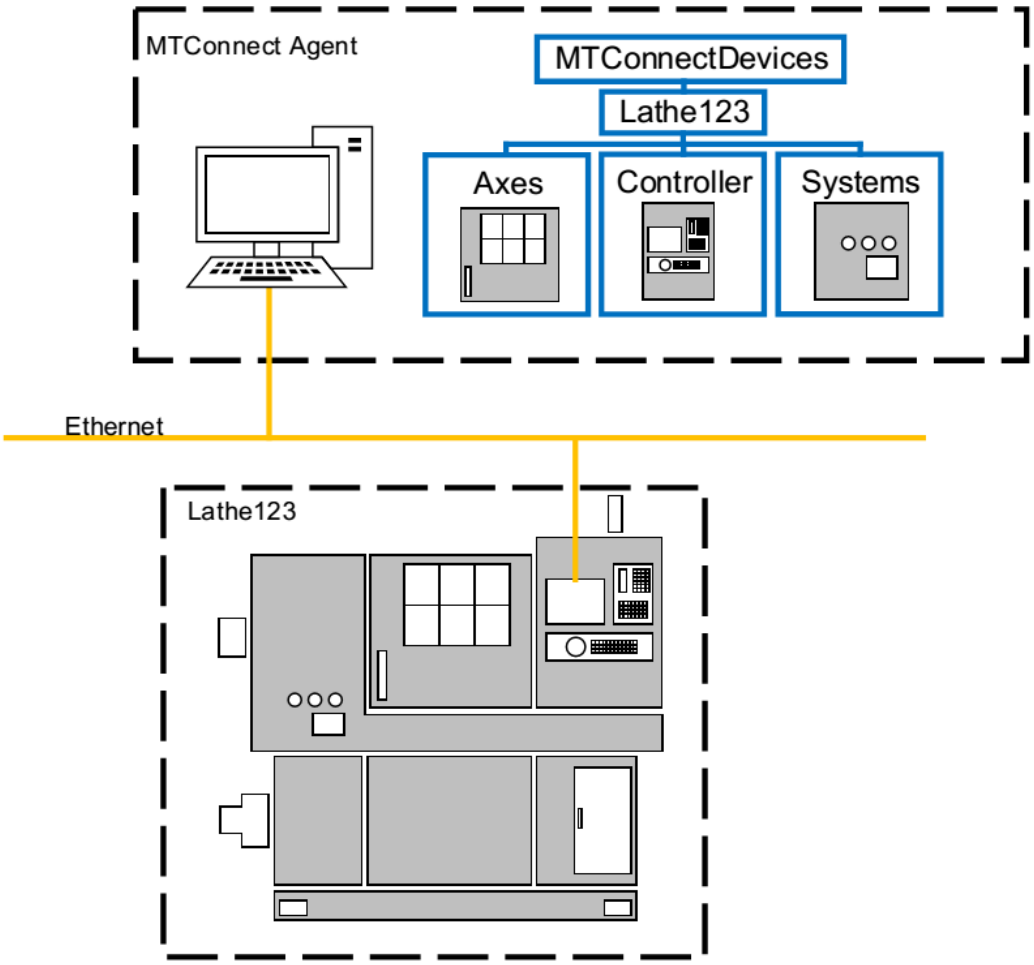

Abbildung 7: MTConnect Informationsmodell im Agent

Die Struktur, die MTConnect mittels XML Schema für Geräte (Devices) vorgibt, kann in das in Abbildung 8 dargestellte Modell übertragen werden, wobei hier nur ein Teil

[1] Siehe: github.com/mtconnect/schema

der nativ implementierten *Component*- und *Subcomponenttypes* gezeigt wird. Außerdem wird in Abbildung 8 nur der physische Teil des Strukturbaumes dargestellt, denn viele der dargestellten Elemente können noch zusätzliche Unterelemente, wie Beschreibungen, Konfigurationsdaten, DataItems oder weitere Komponenten besitzen und des Weiteren auch beschreibende oder identifizierende Attribute beinhalten. Die Daten, die die Maschine liefert, können so pro Datenpunkt (DataItem) der jeweiligen Komponente zugeordnet werden. Die Elemente *Devices*, *Components* und *DataItems* dienen hierbei nur als Container für mehrere Elemente des Types Device bzw. Component oder DataItem.

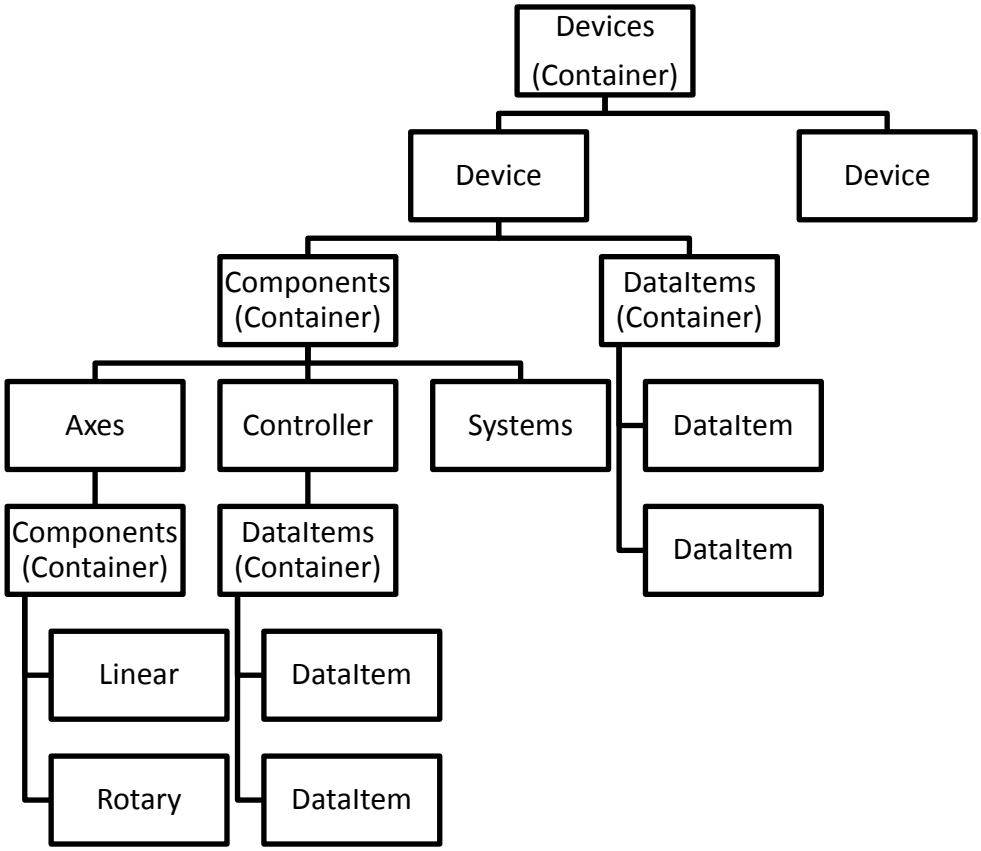

Abbildung 8: Vereinfachter, unvollständiger Gerätestrukturbaum nach [17]

Code-Snippet 5 zeigt, wie diese Struktur im XML Format in einem Agent verwendet werden würde. Dabei steht das Element *MTConnectDevices* als Root Element an der Spitze der Hierarchie. Der Ausschnitt zeigt nur einen kleinen Teil der möglichen Komponenten, denn neben den linearen Achsen (Zeile 13) können z. B. auch rotierende Achsen dargestellt werden und anstatt des Kühlmittelsystems (Zeile 24) kann auch ein elektrisches oder hydraulisches System dargestellt werden.

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <MTConnectDevices xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xmlns:xsd=http://www.w3.org/2001/XMLSchema
      xmlns="urn:mtconnect.org:MTConnectDevices:1.3">
3.    <Header version="1.0" creationTime="2016-02-05T12:07:58.3704691Z" instanceId="1"
        sender="192.168.0.29" bufferSize="131072" assetBufferSize="1024"
        assetCount="0" />
4.    <Devices>
5.      <Device id="dev" name="VMC-3Axis" sampleInterval="10" uuid="000">
6.        <Description manufacturer="SystemInsights"/>
7.          <DataItems>
8.            <DataItem category="EVENT" id="avail" type="AVAILABILITY"/>
9.          </DataItems>
10.       <Components>
11.         <Axes id="ax" name="Axes">
12.           <Components>
13.             <Linear id="x1" name="X">
14.               <DataItems>
15.                 <DataItem category="SAMPLE" id="x2" name="Xact"
                      nativeUnits="MILLIMETER" subType="ACTUAL" type="POSITION"
                      units="MILLIMETER"/>
16.                 <DataItem category="SAMPLE" id="x3" name="Xcom"
                      nativeUnits="MILLIMETER" subType="COMMANDED" type="POSITION"
                      units="MILLIMETER"/>
17.                 <DataItem category="SAMPLE" id="n3" name="Xload"
                      nativeUnits="PERCENT" type="LOAD" units="PERCENT"/>
18.               </DataItems>
19.             </Linear>
20.           </Components>
21.         </Axes>
22.         <Systems id="systems" name="systems">
23.           <Components>
24.             <Coolant id="cool" name="coolant">
25.               <DataItems>
26.                 <DataItem category="CONDITION" id="clow" type="LEVEL"/>
27.                 <DataItem category="CONDITION" id="cpres" type="PRESSURE"/>
28.               </DataItems>
29.             </Coolant>
30.           </Components>
31.         </Systems>
32.       </Components>
33.     </Device>
34.   </Devices>
35. </MTConnectDevices>
```

Code-Snippet 5: MTConnectDevices XML Struktur[2]

Aus dieser Struktur wird auch ersichtlich, dass es neben den in Abbildung 8 gezeigten Komponenten noch eine Vielzahl an zusätzlichen Metadaten in Form von Attributen gibt. Für eine gesamte Liste an möglichen Komponenten und deren spezifischer Attribute müssen entweder der Standard Teil 2 oder das zugehörige XML Schema betrachtet werden. Zu den wichtigsten Attributen zählen der Identifier (*ID*), die Kategorie (*Category*) und der Typ (*Type*) bzw. Subtyp (*Subtype*).

[2] Gekürzte Struktur des offiziellen MTConnect Agent Simulators von http://agent.mtconnect.org/

2.2.2.1 Identifier

In einer MTConnect XML Struktur besitzen alle Elemente einen eindeutigen Schlüssel zur Identifikation. Dieser Wert darf im gesamten Dokument nur ein einziges Mal vorkommen und ist besonders für die DataItems notwendig, da der Agent bei der Kommunikation mit Adaptern pro Datenpunkt jeweils nur einen Zeitstempel, einen Wert und die zugehörige ID erhält und darüber die Werte den Komponenten im Informationsmodell zuordnet. Ebenso können diese Identifier in den Clientanfragen an den Agent verwendet werden, um spezifische Datenpunkte abzufragen [17].

2.2.2.2 Category

Ein weiteres wichtiges Attribut, das jedem DataItem zugewiesen werden muss, ist dessen Kategorie. Diese hat den Zweck Aufschluss über die Art des jeweiligen Datenpunktes zu geben. Hierfür gibt es die drei Kategorien: *Sample*, *Event* und *Condition*. Sample-Werte sind kontinuierlich auftretende Daten, die bei Abfragen jederzeit einen aktuellen und meist variablen Wert liefern können. Diese Werte können entweder den Zustand *Unavailable* oder jeden möglichen Zahlenwert des Datentyps Double annehmen, sofern dieser nicht im Schema begrenzt wurde. Samplewerten muss außerdem mit dem Units-Attribut eine Einheit zugeordnet werden. Das DataItem in Zeile 15 von Code-Snippet 5 zeigt z. B. die aktuelle Position entlang der X-Achse in der Einheit Millimeter an. Events stellen im Gegensatz zu den Sample-Werten diskrete Zustände dar. Das können Statuszustände mit fest vorgegeben Werten sein, wie die Verfügbarkeit der Maschine (Zeile 8), oder einfache Textnachrichten, wie der Name eines aktuell auf der Maschine laufenden Bearbeitungsprogrammes. Zuletzt gibt es die Kategorie Condition, welche die Funktionstüchtigkeit von Komponenten beschreibt. In den Zeilen 26 und 27 wird hiermit z. B. der Zustand des Kühlschmiermittelsystems dargestellt. DataItems dieser Kategorie besitzen neben dem Wert, der Leer oder eine Art von schriftlicher Fehlermeldung sein kann, noch einen Zustand, der entweder *Normal*, *Unavailable*, *Fault* oder *Warning* [17] ist. Da an einer Komponente mehrere Fehlermeldungen zeitgleich auftreten können, können diese DataItems auch gleichzeitig mehrere gültige Werte liefern.

2.2.2.3 Type

Das *Type*-Attribut dient wie die Kategorie einer genaueren Einteilung eines Datenpunktes und muss ebenfalls für jeden Datenpunkt angegeben werden. Es ordnet jedem DataItem seine spezifische Bedeutung, basierend auf den im MTConnect Standard vorgegebenen Datentypen, zu. Die in MTConnect verfügbaren Typen sind von der jeweiligen Kategorie abhängig und hängen im Falle der Sample-Kategorie direkt mit den vorgegebenen Einheiten zusammen. In manchen Fällen kann es neben dem Type-Attribut selbst auch noch ein *SubType*-Attribut geben [17]. Die Datenpunkte in den Zeilen 15 und 16 unterscheiden sich z. B. durch ihren

Subtyp. DataItem x2 stellt die aktuelle Position der Maschinenachse und x3 die von der Steuerung berechnete Position für die Bewegung dar.

Das Type-Attribut ist besonders wichtig, da das Element DataItem an sich nur ein abstraktes Element darstellt, welches vom Agent bei der Datenübertragung an einen Client mit dem jeweiligen Typ ersetzt wird. Die Definition der Typen ist daher nicht Teil des MTConnectDevices Schemas, sondern wird genauer im MTConnectStreams Schema zur Kommunikation behandelt. Code-Snippet 6 zeigt, dass das Type-Attribut im Devices Schema als Zeichenkette mit den in Zeile 8 als regulärer Ausdruck vorgegeben erlaubten Zeichen angegeben wird und nicht in Form einer Liste mit vorgegebenen Typen.

```
1.  <xs:simpleType name="DataItemExtType">
2.    <xs:annotation>
3.      <xs:documentation>
4.        An extension point for data item types
5.      </xs:documentation>
6.    </xs:annotation>
7.    <xs:restriction base="xs:string">
8.      <xs:pattern value="[a-ln-z]:[A-Z_0-9]+"/>
9.    </xs:restriction>
10. </xs:simpleType>
```

Code-Snippet 6: DataItem Extension in MTConnectDevices

2.2.3 Kommunikationsmodell

Teil 3 des MTConnect Standards behandelt das Streams Information Model, mit welchem die Kommunikation zwischen Agent und Client beschrieben wird. Während Teil 2 des Standards für die Strukturierung eines Gerätes oder Adapters nach dem Informationsmodell genutzt wird, ist Teil 3 für die Entwicklung und Implementation von Softwareanwendungen, die Daten über MTConnect abfragen, notwendig. Das Modell beschreibt dabei genau, mit welcher Struktur der MTConnect Agent die angefragten einzelnen Datenwerte an den Client sendet (Abbildung 9).

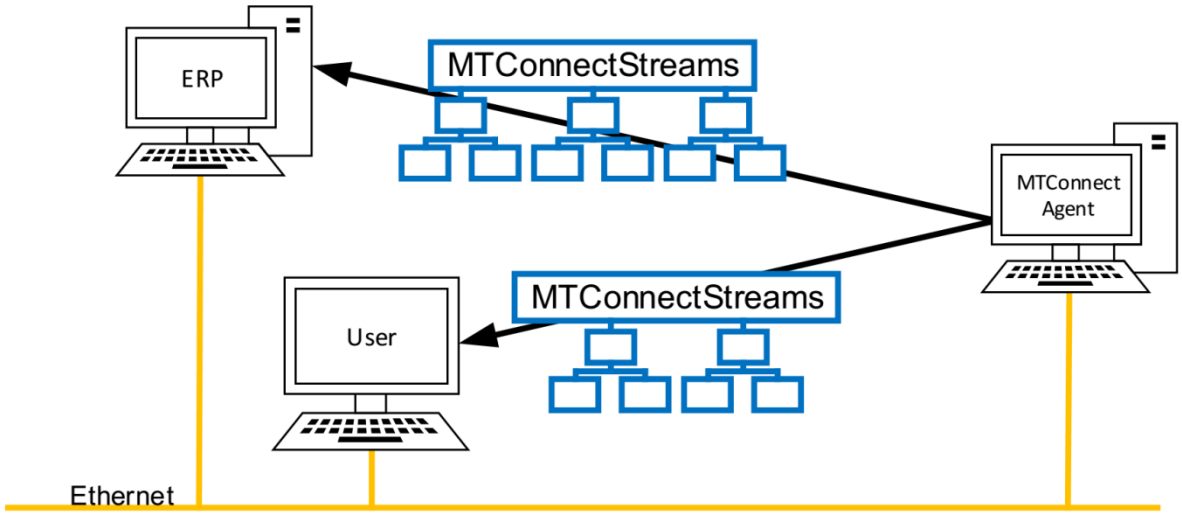

Abbildung 9: Kommunikationsschema

Das oberste Element in dieser Struktur ist *MTConnectStreams*, welches ein Headerelement mit kommunikationsspezifischen Informationen (Sequenznummer und Anzahl der gesendeten Daten) und ein Element namens Streams mit den versendeten Daten enthält. Die Datenwerte werden, entsprechend dem Aufbau in der Gerätedatenstruktur, in die in Abbildung 10 gezeigte Datenstruktur eingefügt. Dabei wird jedes Gerät in einem *DeviceStream* Element, jede Komponente dieses Gerätes in einem untergeordneten *ComponentStream* Element und jedes DataItem, gruppiert nach dessen Kategorie, innerhalb der Komponente dargestellt [18].

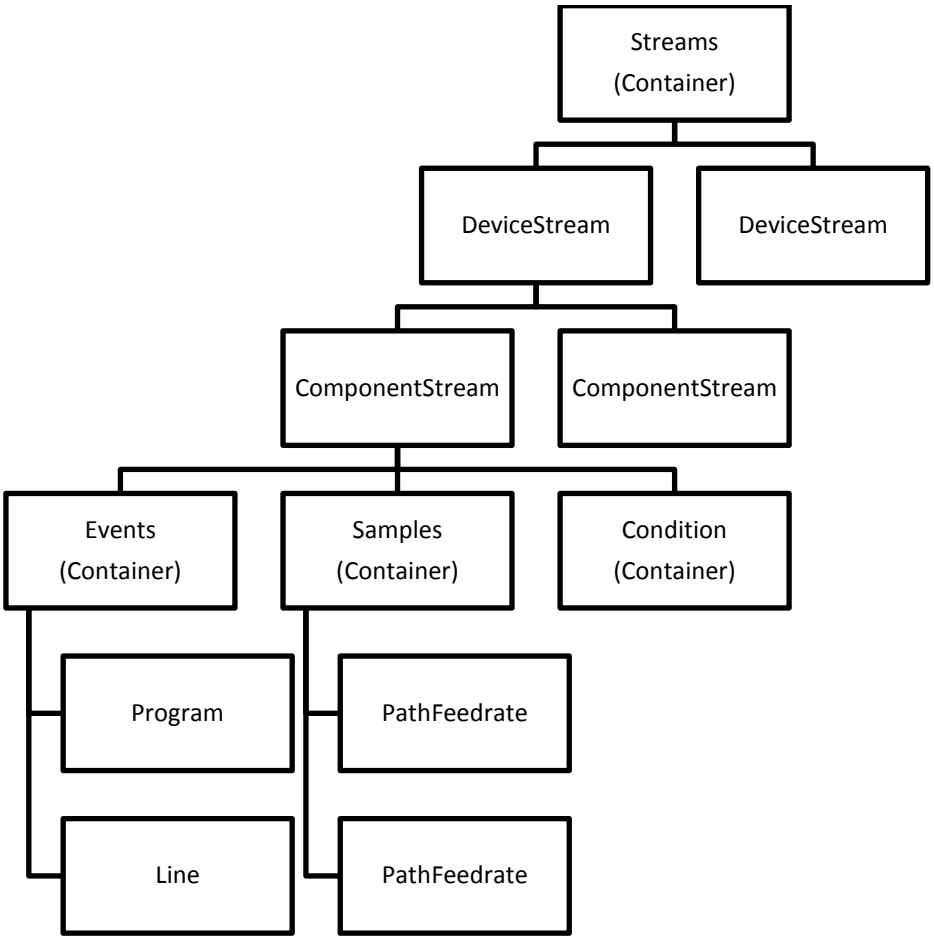

Abbildung 10: Datenstruktur im Streams Modell [18]

Code-Snippet 7 zeigt ein MTConnectStreams Dokument, welches auch in Verbindung mit dem in Zeile 2 referenzierten Stylesheet von einem Webbrowser grafisch oder in Form einer Tabelle, dargestellt werden könnte. Erwähnenswert ist, dass Datenwerte, welche direkt dem Gerät und nicht einer Unterkomponente zugeordnet wurden, ebenso in einem eigenen ComponentStream für das Gerät selbst eingeordnet werden (Zeilen 7 bis 13). Im ComponentStream der Sensorkomponente (Zeilen 14 bis 26) wird die Unterteilung der Datenwerte in die drei Kategorien deutlich. Hierbei sind die Datenwerte der Kategorien Sample und Event nun nicht mehr unter ihrem abstrakten Elementnamen DataItem dargestellt, sondern mit ihrem zugewiesenen Type-Attribut als Element. Falls das Element noch einen zusätzlichen Subtyp besitzt, wird dieser trotzdem nur als Attribut eingefügt. Elemente der Kategorie Condition erhalten ihren Zustand als Elementnamen (Zeile 24), also Normal, Unavailable, Fault oder Warning und ihren Typ bzw. Subtyp als Attribut. Falls das Element einen Wert, wie eine Fehlermeldung besitzt, ist dieser als CDATA enthalten.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <?xml-stylesheet type="text/xsl" href="/styles/Streams.xsl"?>
3.  <MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.3"
      xmlns="urn:mtconnect.org:MTConnectStreams:1.3"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:e="urn:Extension:THD"
      xsi:schemaLocation="urn:Extension:THD ../schemas/ThdExtension.xsd">
4.    <Header creationTime="2016-02-17T10:54:55Z" sender="TTR" instanceId="1455706485"
        version="1.3.0.15" bufferSize="131072" nextSequence="9" firstSequence="1"
        lastSequence="8"/>
5.    <Streams>
6.      <DeviceStream name="MTConnect_PAC" uuid="MTConnect_PAC">
7.        <ComponentStream component="Device" name="MTConnect_PAC"
            componentId="MTConnect_PAC">
8.          <Events>
9.            <AssetChanged dataItemId="MTConnect_PAC_asset_chg"
                timestamp="2016-02-17T10:54:45.280437Z" sequence="1"
                assetType="">UNAVAILABLE</AssetChanged>
10.           <AssetRemoved dataItemId="MTConnect_PAC_asset_rem"
                timestamp="2016-02-17T10:54:45.280437Z" sequence="2"
                assetType="">UNAVAILABLE</AssetRemoved>
11.           <Availability dataItemId="MTConnect_PAC_avail"
                timestamp="2016-02-17T10:54:45.280437Z"
                sequence="3">AVAILABLE</Availability>
12.         </Events>
13.       </ComponentStream>
14.       <ComponentStream component="Sensor" name="SentronPAC4200"
            componentId="PAC_1">
15.         <Samples>
16.           <e:TotalHarmonicDistortion dataItemId="PAC_1_43"
                timestamp="2016-02-17T10:54:45.280437Z" name="THD"
                sequence="4">UNAVAILABLE</e:TotalHarmonicDistortion>
17.           <Wattage dataItemId="PAC_1_65" timestamp="2016-02-17T10:54:45.280437Z"
                name="Total Active Power" sequence="5">UNAVAILABLE</Wattage>
18.           <VoltAmpereReactive dataItemId="PAC_1_67"
                timestamp="2016-02-17T10:54:45.280437Z" name="Total Reactive Power"
                sequence="6">UNAVAILABLE</VoltAmpereReactive>
19.         </Samples>
20.         <Events>
21.           <Availability dataItemId="PAC_1_avail"
                timestamp="2016-02-17T10:54:45.280437Z" name="Availability"
                sequence="7">UNAVAILABLE</Availability>
22.         </Events>
23.         <Condition>
24.           <Unavailable dataItemId="PAC_1_com"
                timestamp="2016-02-17T10:54:45.280437Z" sequence="8"
                type="COMMUNICATIONS"/>
25.         </Condition>
26.       </ComponentStream>
27.     </DeviceStream>
28.   </Streams>
29. </MTConnectStreams>
```

Code-Snippet 7: MTConnectStreams

Bei der Kommunikation wird somit für jeden einzelnen Datenwert ein Element mit den zugehörigen Attributen erzeugt. Der Wert selbst liegt dabei im CDATA Teil des Elementes. Außerdem wird für jeden Datenwert der zugehörige Zeitstempel, dessen ID und dessen eindeutige Sequenznummer innerhalb des Agenten mitgeliefert.

2.2.4 Erweiterung des Schemas

Ein wichtiger Punkt von MTConnect ist die Erweiterbarkeit und Anpassungsfähigkeit des Standards. Da es für verschiedene Maschinen eine Vielzahl von Variablen gibt, kann es vorkommen, dass bestimmte Komponenten, Typen oder Asset-Typen benötigt werden, die nicht nativ durch MTConnect unterstützt werden. Ein Beispiel hierfür ist der Klirrfaktor (englisch: *Total Harmonic Distortion* – THD_R), der das Verhältnis von Oberschwingungen zur Grundschwingung in Prozent misst. Um für den Agent und die verbundenen Clients eine gemeinsame Definition für diesen DataItemtyp zu schaffen, sollte das Schema erweitert werden. Da für MTConnect mehrere Schemata für unterschiedliche Anwendungsbereiche bestehen, muss auch das jeweils entsprechende Schema erweitert werden, mit dem der zu erweiternde Teil beschrieben wird. Die vier von MTConnect vorgegebenen Schemata sind: Assets, Devices, Error und Streams. Wird ein neuer Assettyp hinzugefügt, muss in der Erweiterungsschemadatei das Asset Schema importiert werden. Die Gerätekomponenten dagegen werden im Devices Schema beschrieben. Sollen neue Typen für DataItems ergänzt werden, so muss dabei auf das Streams Schema referenziert werden, da DataItems im Devices Schema nur als abstrakte Klasse behandelt werden. Der Agent ersetzt den abstrakten Namen DataItem erst bei der Kommunikation mit einem Client mit dem jeweiligen Datentyp.

Zur Erweiterung muss die Originalstruktur der MTConnect Schemata beibehalten werden und nur um einen neuen Namespace erweitert werden, wobei der Präfix *m* bereits von MTConnect selbst belegt wird. Außerdem muss MTConnect weiterhin als oberstes Element in der Hierarchie stehen. Unter dem neuen Namespace werden dann die neuen Elemente und deren Zusammenhänge modelliert und diese durch Referenzieren auf Elemente des MTConnect Schemas verknüpft. Der MTConnect C++ Agent, der vom MTConnect Institute veröffentlicht wurde, kann daraufhin so konfiguriert werden, dass Clients per HTTP Anfrage auf die lokalen Schemadateien des Agents und deren Erweiterungen zugreifen können und diese so auslesen bzw. übernehmen können [19].

Das in Code-Snippet 8 gezeigte Schema stellt die Erweiterung des MTConnectStreams Schemas dar, welches hier unter dem Kürzel *mtc* (Zeile 2) referenziert wurde. Die Elemente dieses Schemas entsprechen der Modellierung eines DataItemtyps im originalen MTConnect Schema und können daher für Erweiterungen um neue Datentypen meist großteils übernommen werden. Hinzugefügt werden müssen dann nur die Namensräume (Zeile 2) und das originale Schema (Zeile 4). Die Attribute in den Zeilen 13 und 19 referenzieren auf die Elemente SampleType und Sample aus dem Streamsschema und müssen daher mit dem Bezeichner verlinkt werden. In Zeile 14 wird außerdem die erlaubte Struktur von

Datenwerten beschrieben. Nach Teil 3 des MTConnect Standards [18] stellen Samplewerte immer Kommazahlen oder die Buchstabenfolge UNAVAILABLE dar.

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns="ExtendedNewNamespace" targetNamespace="ExtendedNewNamespace"
      xmlns:mtc="urn:mtconnect.org:MTConnectStreams:1.3">
3.
4.    <xs:import namespace="urn:mtconnect.org:MTConnectStreams:1.3"
        schemaLocation="/Schemas/MTConnectStreams_1.3.xsd"/>
5.
6.    <xs:complexType name="TotalHarmonicDistortionType">
7.      <xs:annotation>
8.        <xs:documentation>
9.          The THD of a component
10.       </xs:documentation>
11.     </xs:annotation>
12.     <xs:simpleContent>
13.       <xs:restriction base="mtc:SampleType">
14.         <xs:pattern value="[+-]?\d+(\.\d+)?([Ee][+-]?\d+)?|UNAVAILABLE"/>
15.       </xs:restriction>
16.     </xs:simpleContent>
17.   </xs:complexType>
18.
19.   <xs:element name="TotalHarmonicDistortion" type="TotalHarmonicDistortionType"
        substitutionGroup="mtc:Sample">
20.     <xs:annotation>
21.       <xs:documentation>
22.         The THD of a component
23.       </xs:documentation>
24.     </xs:annotation>
25.   </xs:element>
26.
27. </xs:schema>
```

Code-Snippet 8: Erweiterung des Schemas um den Type Klirrfaktor (THD_R)

Letztendlich muss die neue Schemadatei noch dem zugehörigen Agent hinzugefügt werden, indem in dessen Konfigurationsdatei auf das Schema verwiesen wird. Clients, die dann auf Daten des MTConnect Agents zugreifen, erhalten in den Attributen des MTConnectStreams Headers den neuen Namensraum und den Ort, an dem sich das Schema befindet (Code-Snippet 7, Zeile 3: xsi:schemaLocation="urn:Extension:THD ../schemas/ThdExtension.xsd") und können so das Element (Code-Snippet 7, Zeile 16) dem Namensraum über das jeweilige Kürzel (hier: *e*) zuordnen.

2.2.5 Requests

Ein für den Endanwender von MTConnect notwendiges Thema ist der Datenzugriff über die MTConnect Schnittstelle. Dieser erfolgt über sogenannte Requests, also Anfragen an den Agent über das Netzwerk.

Dazu muss der Agent mittels eines HTTP GET Requests angefragt werden. Eine solche Anfrage stellt bereits die einfache Eingabe der URI des Agents in einem

Webbrowser dar. Jedoch ist eine genauere Spezifikation möglich, um gezielt gewünschte Daten zu erhalten. Generell gibt es in MTConnect vier Typen von Basisanfragen: *Probe*, *Current*, *Sample* und *Asset*. Diese liefern immer Daten im XML Format zurück [15]. Eine Anfrage besitzt somit immer zumindest die Form „http://URI/Anfragetyp“. Eine Anfrage vom Typ Probe gibt immer die Gerätedatenstruktur des Agents, entsprechend dem Schema MTConnectDevices, zurück. Current und Sample liefern XML Dokumente in der Form von MTConnectStreams und Asset liefert Daten in der Form des MTConnectAssets Schemas bzw. bei Erweiterung um die AssetID die aktuelle XML Struktur des jeweiligen Assets.

Der auf die Daten zugreifende Client kann sich also zuerst über eine Probe-Anfrage über die Struktur des Agents und dessen abgebildete Geräte informieren. Außerdem kann er die Schemadateien und, falls vorhanden, die Erweiterungen des Schemas abfragen. Daraufhin sollte zuerst mit einer Current-Anfrage ein komplettes Abbild des aktuellen Zustands ausgelesen werden, um für jeden Datenpunkt einen Anfangswert zu besitzen, bevor mittels Sample-Anfrage die Werte erfasst werden, die sich geändert haben.

Current liefert dabei immer den letzten bzw. aktuellsten Wert eines Datenpunktes, wobei auch dann noch ein Wert dargestellt wird, wenn der letzte Datenwert mit der zugehörigen Sequenznummer bereits so veraltet ist, dass er aus dem Pufferspeicher des Agents entfernt wurde [15]. Current liefert damit immer einen vollen Datensatz aller DataItems. Sample liefert dagegen eine gewünschte Anzahl an historischen Datenwerten (standardmäßig 100 Stück). In beiden Fällen können dazu noch weitere Parameter spezifiziert werden, anhand derer die Ausgabe des Agents auf bestimmte Werte oder Konstellationen reduziert wird. Es können z. B. bestimmte Komponenten oder Attribute, wie Name und ID, herausgefiltert werden. Für diese Anfragen ergibt sich die erweiterte Syntax:

http://URI/Anfragetyp?Filter1=Wert1&Filter2=Wert2&...

Filter beschreibt dabei die Art des Filters bzw. das zu filternde Kriterium und der Wert gibt den spezifischen Wert für den Filter an. Das erste Kriterium wird dabei mit einem Fragezeichen (?) eingeleitet, weitere Kriterien können mit einem Et-Zeichen (&) angehängt werden. Durch Kombination verschiedener Filter können somit genau spezifizierte Datenpunkte abgefragt werden.

Filter	Wert	Erklärung
Count	Ganzzahl zwischen 1 und der Puffergröße des Agents	Gibt die Anzahl der übertragenen Werte vor (gilt nur für Sampleanfragen, Default = 100)
From	Sequenznummer	Gibt die Sequenznummer des ersten DataItems vor (nur bei Sampleanfragen)
Path	//Element1//Element2//…	Gibt alle DataItems in einer Struktur, die dem Pfad (XPath) entsprechen, an
At	Sequenznummer	Gibt die geltenden Zustände zum Zeitpunkt einer bestimmten Sequenznummer zurück (nur bei Currentanfragen) [15]
Interval	Ganzzahl in Millisekunden (Zahl von 0 bis 10000) [15]	Der Agent sendet kontinuierlich im Abstand des Intervalls Daten (Streaming - darf nicht mit dem Filter *At* verwendet werden) [15] *Interval=0* ermöglicht eine augenblickliche Weiterleitung der Daten an einen Client

Tabelle 1: Filter für MTConnect Requests

Die in Tabelle 1 dargestellten Filter *Count*, *From* und *At* dienen der Spezifizierung der Position und Menge der gesuchten Daten innerhalb des Pufferspeichers. Durch die notwendige Angabe einer Sequenznummer bei den Filtern At und From kann es zu einem *Out of Range* Fehler (entsprechend dem MTConnectError Schema) kommen, falls sich der Wert mit der Sequenznummer nicht mehr im Pufferspeicher befindet oder noch nicht existiert. Der Filter *Path* dagegen baut auf der im Agent definierten Struktur auf und hilft dabei gezielt Werte bestimmter Datenpunkte nach Komponente oder Device herauszufiltern. Der Wert des Filters besteht dabei aus einem Pfad von Strukturelementen, getrennt mit zwei Schrägstrichen, nach dem *XPath* Standard. Der Pfad *//Axes//Linear* würde z. B. alle DataItems eines Agents liefern, die einer Subkomponente *Linear* der Komponente *Axes* angehören. Um einen Pfad genauer zu spezifizieren, können Strukturelemente zusätzlich nach deren Attributen gefiltert werden. Dazu wird das Attribut direkt nach dem Strukturelement in eckigen Klammern und mit einem At-Zeichen (@) angefügt. Hierbei können auch mehrere Attribute in einer Abfrage logisch verknüpft werden:

```
//Device[@name=“Device1“]//Axes//Linear//DataItem[@type=“POSITION“ and
@subType=“ACTUAL“]
```

Mit diesem Pfad wird z. B. auf DataItems der Komponente Axes und Subkomponente Linear verwiesen, die einem bestimmten Gerät angehören und den geforderten Typ und Subtyp besitzen. Bei der Verwendung des Pathfilters muss für den Wert des Filters zwingend auf die Groß- und Kleinschreibung von Elementen, Attributen und

Attributwerten geachtet werden, da der Agent sonst mit einer Invalid Path Exception antwortet.

2.2.6 Assets

Ein Asset ist die Repräsentation einer für den Produktionsprozess notwendigen Entität, in Form einer XML Struktur [14]. Da ein Asset im MTConnect Standard nur als eine Entität definiert ist, muss es sich hierbei nicht zwingend um einen physischen Gegenstand handeln. Ein Asset könnte damit auch eine immaterielle Entität sein. Der MTConnect Standard enthält zum aktuellen Zeitpunkt nur Schemadefinitionen für die beiden Assets *CuttingTool* und *CuttingToolArchetype* [20]. CuttingTool repräsentiert ein physisch vorhandenes Schneidwerkzeug einer Werkzeugmaschine mit aktuellen Maßen und Zuständen. Ein Archetyp stellt damit ein ideales Werkzeug dar, auf welches das CuttingTool als Instanz des Archetyps referenziert werden kann. Es handelt sich dabei folglich um ein immaterielles Asset. Der Standard kann jedoch problemlos um weitere Assets erweitert werden, da der MTConnect Agent auch ohne Schemadefinition jede Art von XML Struktur als Asset akzeptiert.

Assets besitzen drei notwendige Attribute: eine *AssetID* zur global eindeutigen Identifizierung dieser Instanz eines Assets, einen *AssetType*, der die Art des Assets kennzeichnet, wie CuttingTool, und eine *DeviceUuid* (UUID = Universally Unique Identifier), mit der das Asset einem bestimmten Gerät zugeordnet wird.

Wenn nun eine Maschine ein Asset erhält, welches beispielsweise die AssetID als Barcode oder in einem RFID-Chip (Radio Frequency Identification) enthält, kann diese Maschine einen oder mehrere Agents im Netzwerk nach dieser ID durchsuchen. Daraus kann dann die aktuelle Struktur des Assets im XML Format eingelesen, verwendet, upgedatet, entfernt oder einem neuen Device zugeordnet werden.

2.2.7 Interfaces

In der Version 1.3.0 erhielt der MTConnect Standard im September 2014 mit Teil 3.1 eine Neuerung: die *Interfaces*. Interfaces werden im Informationsmodell wie Komponenten behandelt und stellen die physischen oder logischen Schnittstellen von Geräten dar. Diese sind bisher auf die Interaktion von zwei Geräten beschränkt, sollen aber zukünftig auch die Interaktion mit mehreren Geräten erlauben [21].

Diese Schnittstelle stellt bei MTConnect einen Zustandautomaten dar, bei der jedes Gerät genau definierte Zustände besitzt und bestimmte Funktionen bzw. Bedingungen, um diesen Zustand zu erreichen. Eine Maschine besitzt dabei eine Komponente, die die Interaktion mit anderen Maschinen erlaubt und fest definierte Zustände annehmen kann. Eine solche Komponente ist z. B. die Schutztür (*DOOR*) einer Maschine, die die Zustände *OPEN*, *UNLATCHED* und *CLOSED* annehmen kann. Diese Komponente wird auch im Informationsmodell wie eine normale Komponente modelliert und enthält dabei ein DataItem (*DOOR_STATE*), das den aktuellen Zustand angibt (Code-Snippet 9). Zusätzlich benötigt die Maschine eine Modellierung der Funktionen, die eine bestimmte Zustandsänderung herbeiführen. Dafür enthält die Maschine eine Komponente Interfaces (Abbildung 11), die als Subkomponenten alle angebotenen Schnittstellen der Maschine umfasst, u. a. auch das Interface der Schutztür *DoorInterface*.

```
1.  <Door id="door">
2.    <DataItems>
3.      <DataItem category="EVENT" id="d_state" type="DOOR_STATE"/>
4.    </DataItems>
5.  </Door>
```

Code-Snippet 9: DoorState

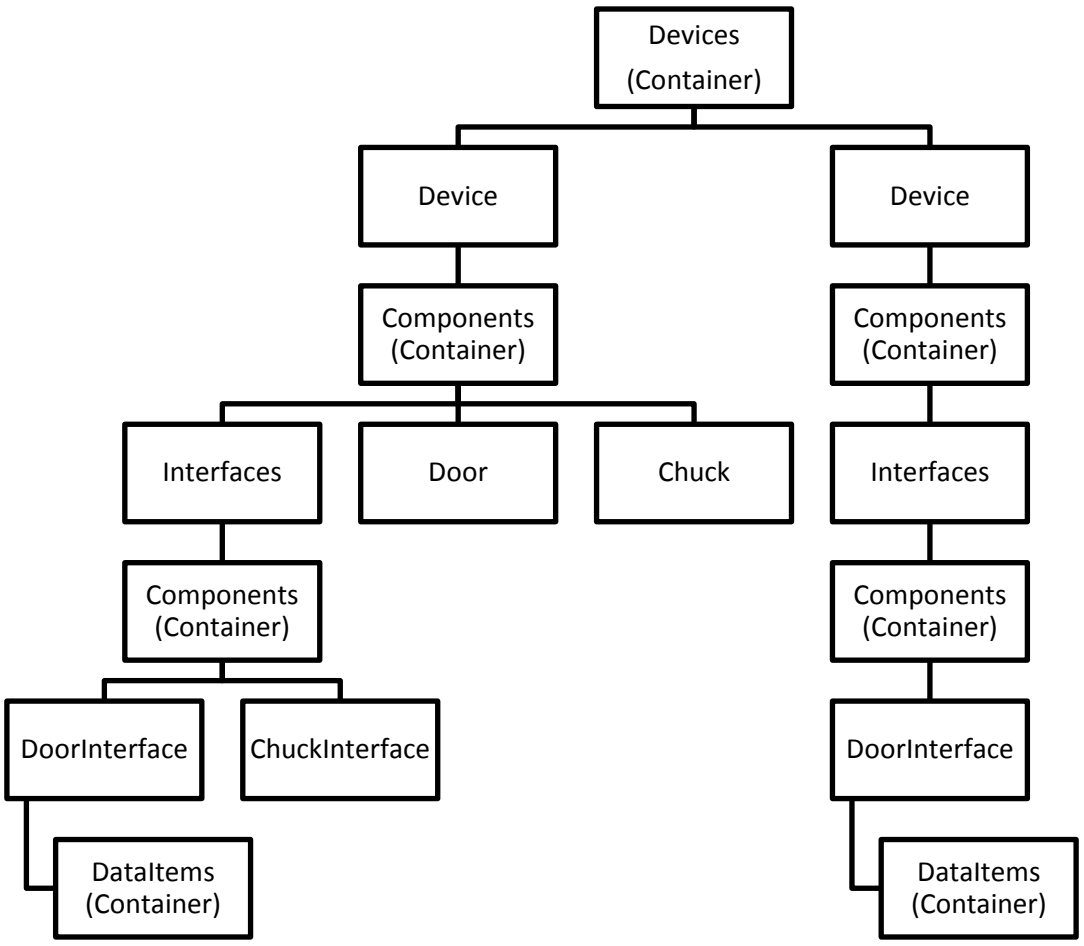

Abbildung 11: Interfaces im Informationsmodell

Dieses Interface enthält für die beiden Funktionen *OPEN_DOOR* und *CLOSE_DOOR*, die von der Maschine ausgeführt werden können, um die Zustände zu ändern, jeweils ein DataItem (Code-Snippet 10).

```
1.  <DoorInterface id="doorInterfaceRes">
2.    <DataItems>
3.      <DataItem category="EVENT" id="od_res" type="OPEN_DOOR" subType="RESPONSE"/>
4.      <DataItem category="EVENT" id="cd_res" type="CLOSE_DOOR" subType="RESPONSE"/>
5.    </DataItems>
6.  </DoorInterface>
```

Code-Snippet 10: DoorInterface Maschinenseite

Damit nun andere Geräte die Funktionen verwenden können, um die Zustände zu verändern, wird bei den Funktionen zwischen einer *Request*- und einer *Response*-Seite unterschieden, die als Subtyp angegeben wird. Die Request-Seite (Code-Snippet 11) wird als Interface des Gerätes implementiert, das die Schnittstelle nutzen möchte, die Responseseite stellt das Gerät dar, welches die Schnittstelle als Komponente besitzt und deren Zustände steuert.

```
1.  <DoorInterface id="doorInterfaceReq">
2.    <DataItems>
3.      <DataItem category="EVENT" id="od_req" type="OPEN_DOOR" subType="REQUEST"/>
4.      <DataItem category="EVENT" id="cd_req" type="CLOSE_DOOR" subType="REQUEST"/>
5.    </DataItems>
6.  </DoorInterface>
```

Code-Snippet 11: DoorInterface Requestseite

Abbildung 12 zeigt, wie die Kommunikation über die Schnittstelle abläuft. Dabei fragt ein Gerät den jeweiligen Zustand des anderen Gerätes über den MTConnect Agent ab. Die Maschine fragt also die Requests des Gerätes und das anfragende Gerät die Antwort der Maschine und den Zustand der Schnittstelle ab. Die Kommunikation beschränkt sich dabei ausschließlich auf Zustandsabfragen an den oder die Agents, die das jeweilige DataItem enthalten. Die Aktualisierung ihrer DataItems im Agent übernehmen die Geräte auf die gleiche Weise, wie sie ihre sonstigen DataItems aktualisieren, also über ihre Adapterschnittstelle. Für die Abfragen der Datenpunkte sollte außerdem nicht die Polling Methode verwendet werden, sondern die Streaming Methode mit einem Intervall von 0 (Tabelle 1):

http://URI/sample?interval=0&path=//DataItem[@id="od_req" or @id="cd_req"]

Dabei sollte der Path-Filter für das Element DataItem mit einer logischen Verknüpfung der DataItemIDs verwendet werden, damit der Umfang des Streams und damit die Anzahl der DataItems des Streams beschränkt wird. Diese Art der

Kommunikation erlaubt eine augenblickliche Weiterleitung von Zustandsänderungen durch den Agent (Streaming – siehe Tabelle 1).

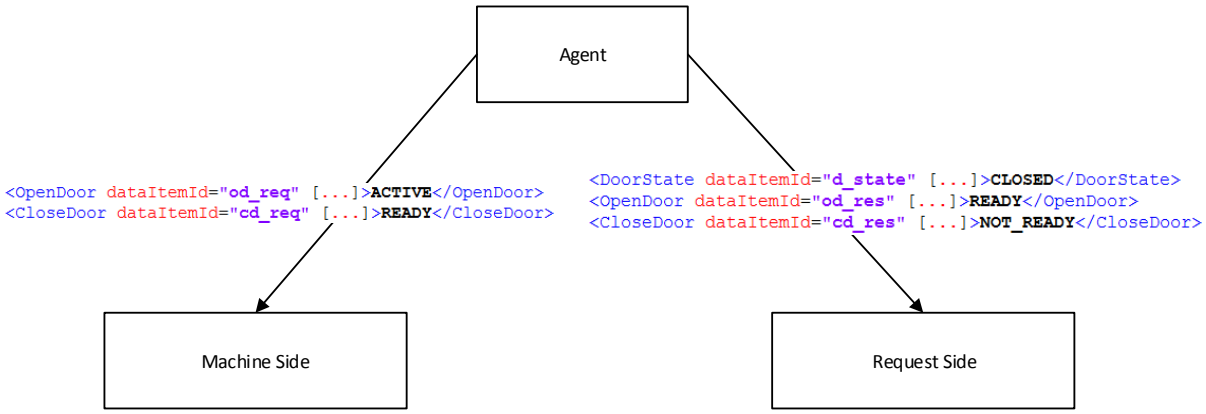

Abbildung 12: Kommunikation über Interfaces

Die Zustände der Funktionen bei der Kommunikation sind von der jeweiligen Seite abhängig. Abgesehen von Fehlerzuständen kann die Requestseite die Zustände *NOT_READY*, *READY* und *ACTIVE* annehmen, wobei ACTIVE die Forderung zur Ausführung dieser Funktion darstellt. Auf der Responseseite stellt dieser Zustand die Ausführung der Funktion dar, welche mit dem Zustand *COMPLETE* abgeschlossen wird (Abbildung 13). Nach vollendeter Aufgabe nehmen beide Seiten wieder den Zustand READY bzw. NOT_READY, wenn die Funktion nicht verwendet werden kann, an.

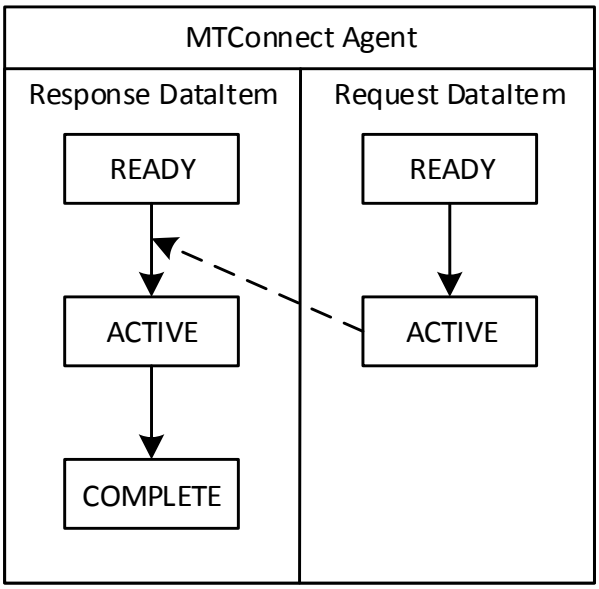

Abbildung 13: Kommunikationsabfolge nach [21]

Meist gibt es aber nicht nur auf einer Seite eine Schnittstelle, wie eine Schutztür, die genutzt wird, sondern auch auf der gegenüberliegenden Seite, wie ein zweites Gerät, das eine Interaktion über die geöffnete Tür ausführt. Dabei ist dann die das Geräte auf der Türseite der Requester der Aktion und das zweite Gerät der ausführende

Teil. Diese Schnittstelle ist notwendig, damit die Türseite weiß, wann eine Interaktion über die Tür ausgeführt wird und wann die Tür wieder geschlossen werden darf. Dadurch wird mit Sicherheit gewährleistet, dass die Tür nicht geschlossen wird, während ein Prozess ausgeführt wird.

MTConnect Interfaces bieten damit vielversprechende Aussichten für zukünftige Anwendungen und schließen eine elementare Lücke des Standards, die nach einer standardisierten Interaktion zwischen Geräten verlangt.

3 MTConnect Sensor- und Steuerungsintegration

Dieser Abschnitt behandelt die praktische Integration des MTConnect Standards innerhalb der Pilotfabrik der Technischen Universität Wien in Aspern zur Erfassung der Steuerungs- und elektrischen Leistungsdaten einer Drehmaschine. Dazu erfolgt zuerst eine Beschreibung des Systemaufbaus in Kapitel 3.1, in dem die einzelnen Komponenten und deren Kommunikationsprotokolle beschrieben werden. Danach werden die mit diesem Aufbau verfolgten Ziele definiert und die notwendigen Parameter des Systems festgelegt. Kapitel 3.3 behandelt die Verwendung und Konfiguration des offiziellen MTConnect C++ Agents (Version 1.3.0.0) [22] zur Kommunikation mit dem Adapter der Drehmaschine und der Sensor-Adapterschnittstelle für die zusätzlichen Messgeräte. Daraufhin geht es in 3.4 um die Sensorintegration durch Programmierung einer Adapterschnittstelle zwischen den Modbus Messgeräten und dem MTConnect Agent und in 3.5 um die Steuerungsintegration der Drehmaschine mittels Adaptersoftware.

3.1 Komponenten und Aufbau

Das gesamte System besteht aus den folgenden Komponenten:

- Emco Maxxturn 45 Drehmaschine mit Siemens Sinumerik 840D Solution Line auf der Numerical Control Unit (NCU) und MTConnect Adapter
- Netzwerkrechner mit MTConnect Agent und eigens programmierter MTConnect Adaptersoftware (MTConnectPAC) zur Integration der Messgeräte
- Siemens Sentron PAC 4200 Messgeräte (4 Stück) mit Modbus TCP Kommunikationsschnittstelle

Die Daten aus den MTConnect Adaptern (Drehmaschine und Messgeräte) sollen hierbei in einem gemeinsamen Agent auf dem Netzwerkrechner verknüpft werden, um die gemessenen Daten gemeinsam in einem Informationsmodell darzustellen. Die physische Vernetzung der Komponenten erfolgt dabei ausschließlich über die bestehende Ethernet Netzwerkstruktur und erfordert keine zusätzlichen Komponenten. Auch die Messgeräte können mit Modbus TCP/IP über dieses Netzwerk mit der zugehörigen Adaptersoftware auf dem Rechner kommunizieren. Durch die Erfassung des elektrischen Stroms über Stromzangen und der jeweiligen Spannungen über Messleitungen ermitteln die Messgeräte u. a. die elektrischen Wirkleistungen für die einzelnen Komponenten (Hydraulik-, Kühlschmierstoff-, Antriebs- und Gesamtsystem) der Drehmaschine (Abbildung 14).

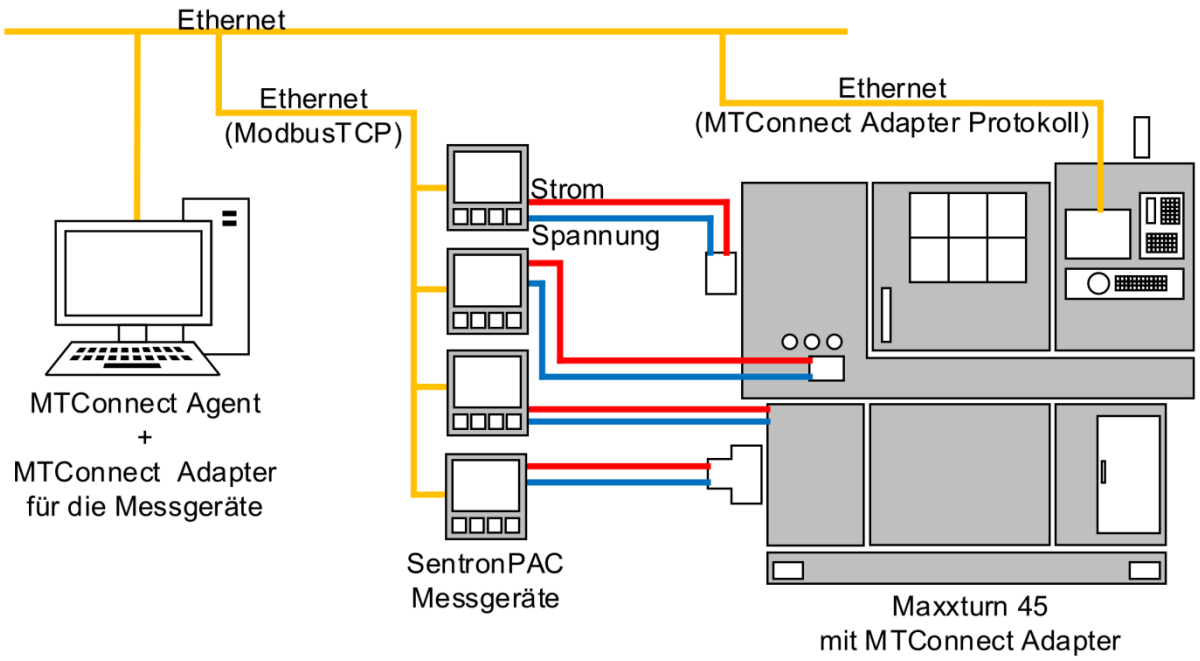

Abbildung 14: Zielsystemaufbau

3.2 Ziele

Das Ziel der Arbeit ist somit die Integration der Steuerungsdaten der Drehmaschine und der Messwerte der Messgeräte in einem gemeinsamen MTConnect Agent. Die Daten verschiedener Komponenten sollen dazu parallel ausgelesen und dem Agent zur Verfügung gestellt werden, damit Steuerungs- und Sensordaten einander zeitlich zugeordnet werden können.

Zu den gesuchten Daten der Drehmaschine zählen Achsenpositionen, Spindeldaten, wie Drehzahlen, und Daten der Steuerung, wie Programme und Programmzeilen. Von den Messgeräten sind dazu nur die jeweiligen Wirkleistungen notwendig, es sollen aber auch andere Messwerte ausgelesen werden können.

Für Adapter und Agent auf dem Netzwerkrechner soll eine gemeinsame grafische Anwendung programmiert werden, in der Agent und Adapter auf einfache und anwenderfreundliche Weise konfiguriert und gestartet werden können. Außerdem müssen Konfigurationen gesichert und wieder geladen werden können, um kurzfristig Änderungen vornehmen zu können. Diese Konfigurationen können innerhalb und außerhalb des Programmes erstellt werden. Da die Anwendung längere Zeit laufen soll, sollte die Auslastung des Rechners nur gering sein. Außerdem darf ein Ausfall einzelner Messgeräte keine Auswirkung auf die Funktionsfähigkeit des Programmes oder anderer Messzyklen haben und eine Rückmeldung über die Verfügbarkeit von Daten an den Agent senden. Die zusätzliche Einbindung des MTConnect C++ Agenten muss außerdem eine sichere automatische Konfiguration der Datenstruktur

im Agenten, ohne Überschreiben der Datenstrukturen anderer Geräte im Informationsmodell, erlauben. Der verwendete MTConnect Standard entspricht der aktuellen Version 1.3.1. Außerdem muss eine einfache Installation der Software ermöglicht werden.

3.3 MTConnect C++ Agent

Der verwendete Agent ist der offizielle MTConnect C++ Agent [22], welcher auch als vorkompilierte Version vorhanden ist. Der Agent an sich ist eine Konsolenanwendung, welche auf Basis einer Konfigurationsdatei und einer XML Datei konfiguriert werden kann. Die XML Datei stellt dabei eine MTConnectDevices Struktur, basierend auf Teil 2 des MTConnect Standards, dar, mit der der Aufbau der verbundenen Geräte nach dem Informationsmodell beschrieben wird (Code-Snippet 5). Die Konfigurationsdatei dient der Einstellung von Parametern, wie z. B. der Größe des Pufferspeichers oder der Verknüpfung von Adaptern bzw. Geräten, welche mit ihrem Namen, ihrer IP und der Portnummer angegeben werden müssen. Außerdem müssen hier Dateien referenziert werden, mit denen ein Schema des MTConnect Standards erweitert wird.

Der Agent kann von der Konsole aus mit mehreren Parametern gestartet werden. Der Parameter run startet den Agent in der Konsole, ohne dass dieser weitere Rückgabewerte oder Informationen in der Konsole liefert. Mit dem Parameter debug liefert der Agent ständig Rückmeldungen zum Verbindungsstatus mit Adaptern und HTTP Anfragen von Clients, was beim Finden von Problemen helfen kann. Der Agent kann neben der Konsolenanwendung auch als Windows Systemdienst installiert und ausgeführt werden.

Sollten mehrere Agents auf demselben Rechner laufen, dann sollten diesen unterschiedliche Konfigurationsdateien mit unterschiedlichen Ports zugewiesen werden. Zur Unterscheidung der Prozesse kann für die Nutzung der Konsolenanwendung die Ausführungsdatei umbenannt werden. Bei Verwendung als Systemdienst kann in der Konfigurationsdatei der Parameter ServiceName angegeben werden.

3.4 Adapterprogrammierung zur Leistungsmessung

Wie bereits in Kapitel 3.1 und 3.2 geschildert, sollen die SentronPAC Messgeräte von einem programmierten Adapter ausgelesen und dem Agent zur Verfügung gestellt werden. Außerdem soll die Anwendung vom/von der NutzerIn konfiguriert werden können, damit dieser/diese die von den Messgeräten gewünschten Datenpunkte bereitstellt. Das Programm soll die Datenstruktur damit automatisch im MTConnect Agent anpassen und so die Bedienung vereinfachen. Die Programmierung erfolgt als Windows Forms Anwendung in der Programmiersprache C# innerhalb der .NET 4.5.2 Programmierumgebung und basiert damit, wie auch die Programmiersprache Java, auf dem Konzept der objektorientierten Programmierung, mit der Softwarekomponenten in Klassen mit Attributen und Methoden eingeteilt und damit strukturiert werden können.

3.4.1 Aufbau

Die Programmstruktur basiert auf dem Drei-Schichten-Architekturmodell mit dem die verschiedenen Programmkomponenten nach ihren Aufgaben in drei Schichten gegliedert werden. Durch diese Trennung lassen sich Komponenten der einzelnen Schichten leichter erweitern oder austauschen. Anwendungen werden demnach in die Bestandteile Benutzeroberfläche, Fachkonzept und Datenhaltung aufgeteilt [23]. Die Schicht Fachkonzept enthält dabei die eigentliche Programmlogik (Business Logic). Die Benutzeroberfläche (Presentation Layer) stellt die Schnittstelle zwischen Anwender und Programmlogik dar. Die Datenhaltung (Data Layer) dient dem Datenzugriff auf Datenbanken, Server oder einfache Dateien.

Abbildung 15 stellt die Programmstruktur als vereinfachtes Klassendiagramm basierend auf der Modelliersprache UML (Unified Modeling Language) und dem Drei-Schichten-Architekturmodell dar.

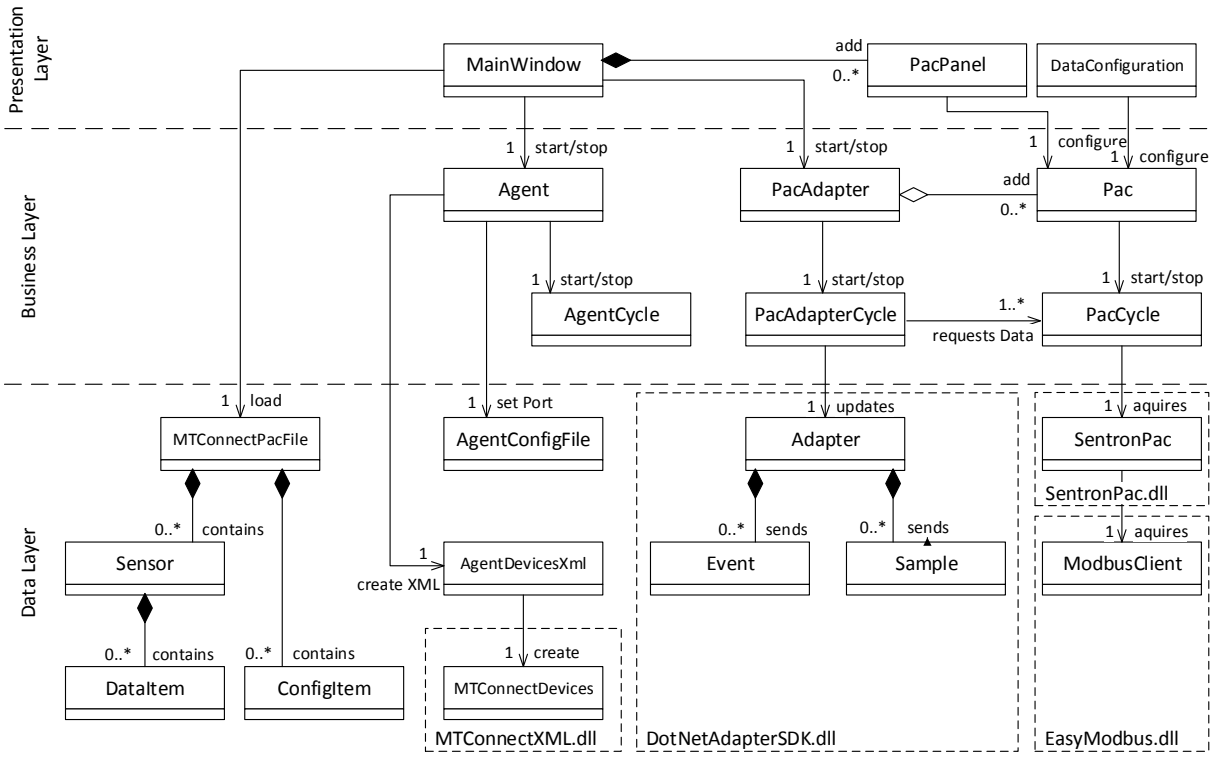

Abbildung 15: UML Klassendiagramm mit Drei-Schichten-Architekturmodell

Hauptausgangspunkt der Anwendung ist das Standardfenster *MainWindow*, in dem der/die AnwenderIn die Parameter des Programmes konfigurieren kann. Diese Anwendung enthält ein Objekt der Klasse *Agent*, welches die Integration des MTConnect C++ Agents in die Anwendung darstellt, ein Objekt der Klasse *PacAdapter*, welches sich um die Integration des MTConnect Adapters mit den einzelnen *Pac* Messgeräten kümmert und ein Objekt der Klasse *MTConnectPacFile*. MTConnectPacFile enthält die konfigurierten Parameter der Anwendung, welche als Datei gespeichert werden und als Konfiguration geladen werden können.

Die Klassen *AgentCycle*, *PacAdapterCycle* und *PacCycle* stellen Threads dar. Diese Threads sind einzelne Prozesse, welche eine gleichzeitige und asynchrone Bearbeitung der Aufgaben der einzelnen Komponenten erlauben und so die Stabilität des Programmes gewährleisten. Somit sind die einzelnen Prozesse von Agent, Adapter und den Messgeräten voneinander unabhängig.

Die Komponente PacAdapter verwendet zur Kommunikation die DotNetAdapterSDK [24], welche vom MTConnect Institute unter der Apache 2 License zur freien Verwendung bereitgestellt ist (Kapitel 3.4.8). Zur Kommunikation mit den einzelnen Messgeräten über ModbusTCP wird die EasyModbus [25] Softwarebibliothek verwendet, die unter der GNU General Public License freigegeben ist. Hierzu wurde zusätzlich als Schnittstelle die SentronPac Bibliothek programmiert, um den spezifischen Zugriff auf die Messgeräte zu vereinfachen (Kapitel 3.4.7). Außerdem

wurde die MTConnectXML Bibliothek programmiert, um eine Schnittstelle zur Konfiguration der MTConnect XML Dateien zu schaffen (Kapitel 3.4.6).

3.4.2 Benutzeroberfläche

Der Presentation Layer des Programmes besteht aus den zwei Windows Forms [26] MainWindow und DataConfiguration. MainWindow ist das eigentliche Anwendungsfenster, mit dem der Anwender sowohl den MTConnect Adapter konfigurieren, als auch Adapter und Agent starten und stoppen kann (Abbildung 16). MainWindow enthält dabei eine Komposition der Klasse PacPanel. Das bedeutet, dass ein Objekt der Klasse MainWindow mehrere Objekte der Klasse PacPanel enthalten kann.

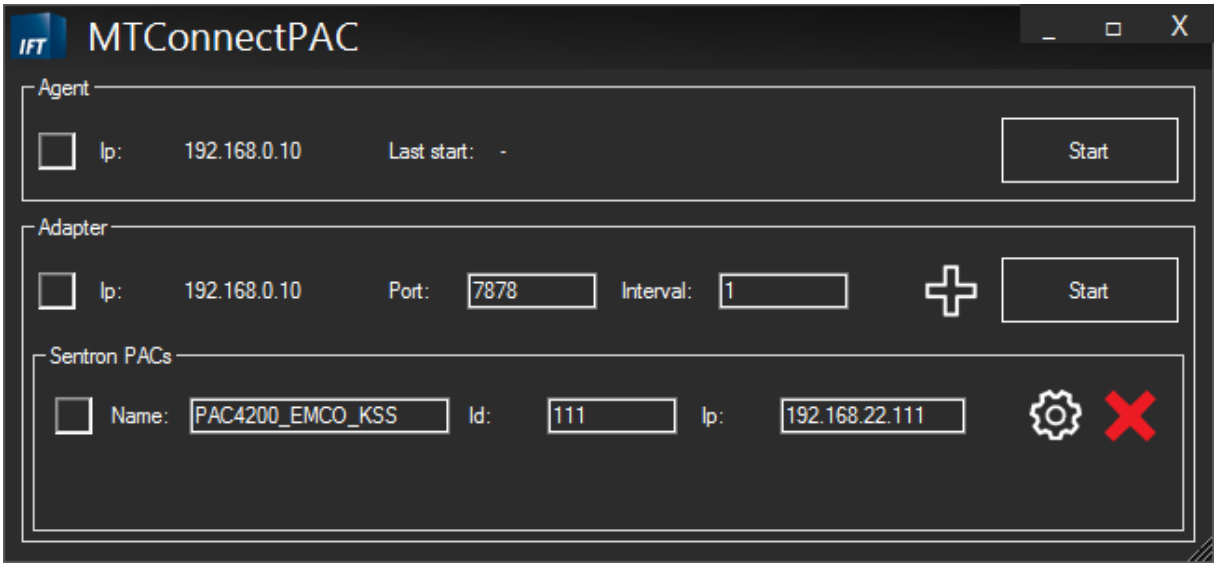

Abbildung 16: MainWindow

PacPanel repräsentiert die Konfigurationsoberfläche eines einzelnen SentronPAC Messgerätes, welches dem Hauptfenster über einen Klick auf das Plussymbol in Abbildung 16 hinzugefügt werden kann. PacPanel (Abbildung 17) ist eine von der Windows Forms Klasse Panel [27] abgeleitete Klasse und kann damit als Container für mehrere Objekte dienen. Ein Klick auf das Kreuzsymbol entfernt das Containerelement wieder. Das Zahnradsymbol ruft das DataConfiguration Fenster auf.

Abbildung 17: PacPanel

DataConfiguration ermöglicht dem/der BenutzerIn die Konfiguration der Datenpunkte der einzelnen Messgeräte. Wie in Abbildung 18 dargestellt, kann der/die AnwenderIn einen Typ aus der Liste aller möglichen Datenpunkte des Messgerätes wählen und

diesem einen eigenen Namen zuordnen, welcher dann im MTConnect Gerätemodell verwendet wird. Änderungen können dann entweder gespeichert oder verworfen werden.

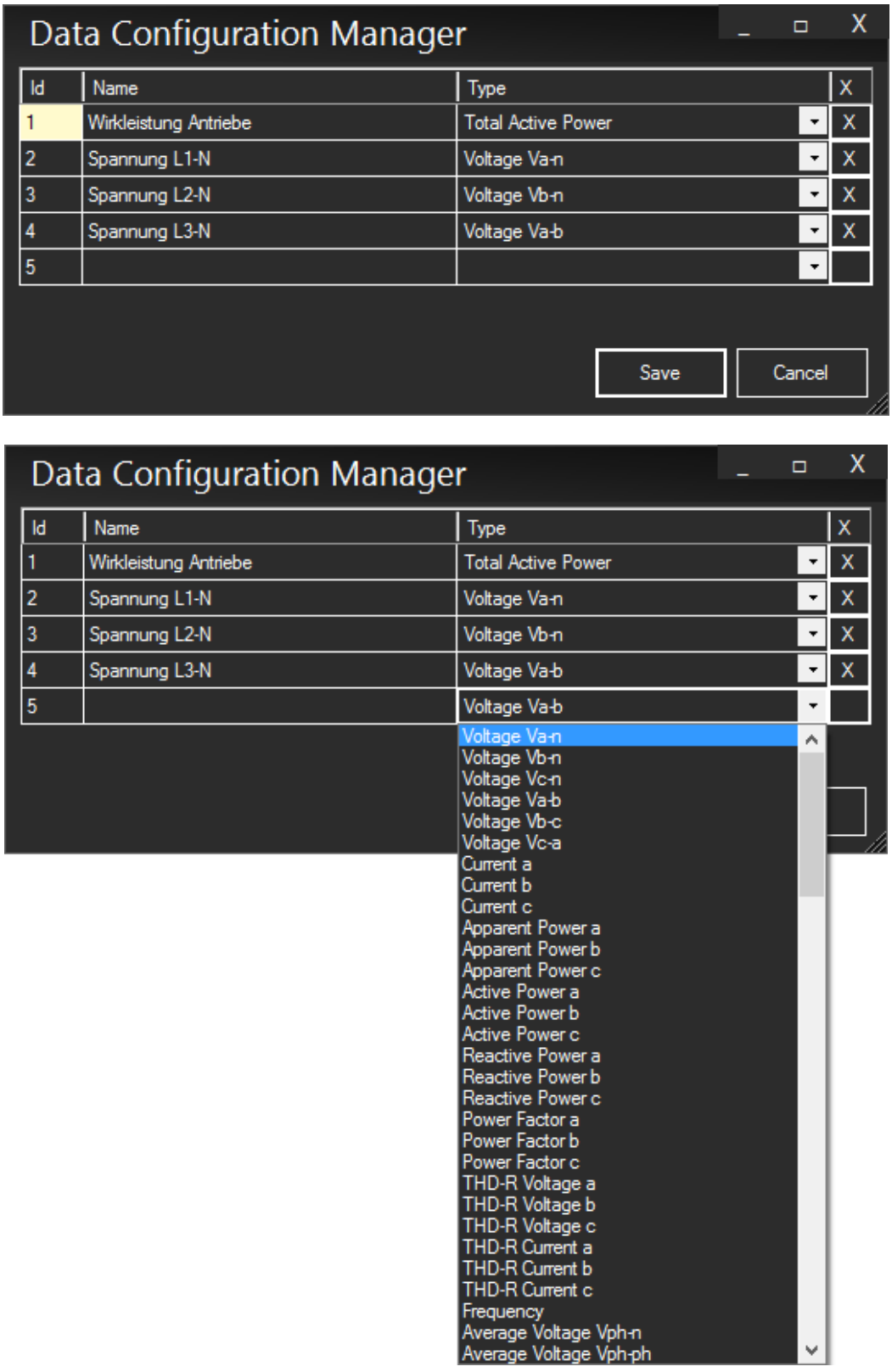

Abbildung 18: DataConfiguration

3.4.3 PacAdapter

Die Klasse PacAdapter enthält, wie in Abbildung 19 dargestellt, ein PacAdapterCycle Objekt mit dem zugehörigen Thread und eine Komposition der Klasse Pac. Die Klasse Pac enthält ein PacCycle Objekt mit dem zugehörigen Thread und hat zusätzlich Zugriff auf die zum Messgerät gehörenden Elemente der Benutzeroberfläche (PacPanel und DataConfiguration), um die dort konfigurierten Parameter, wie IP Adresse und Datenpunkte, einzulesen.

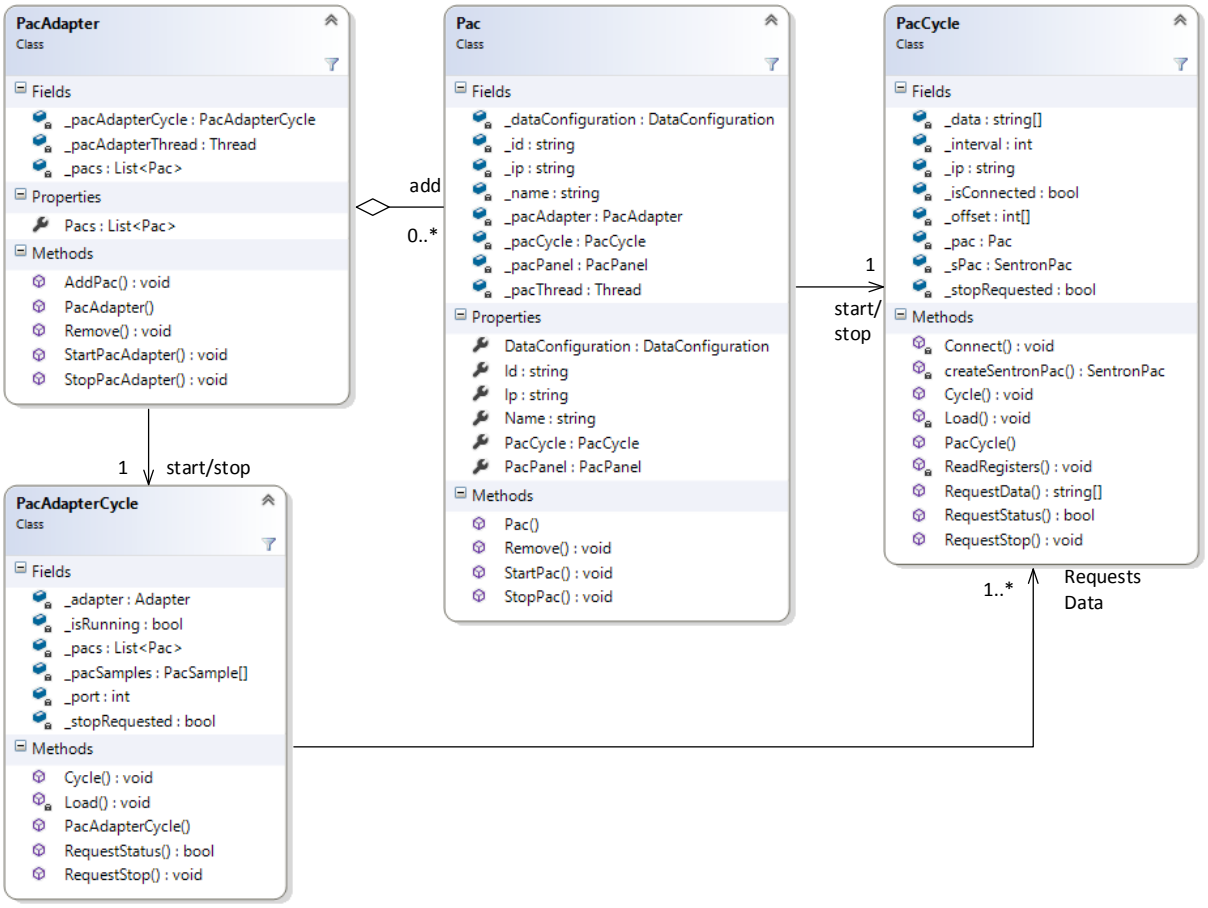

Abbildung 19: PacAdapter und zugehörige Subklassen

Basierend auf dem MTConnect Standard stellt die Klasse PacAdapter damit einen MTConnect Adapter dar, der eine Anzahl von Messgeräten enthält. Diese Messgeräte enthalten wiederum jeweils eine Anzahl an Datenpunkten, die vom Nutzer gewählt werden können. Der Adapter stellt dabei nach dem Geräteschema von MTConnect ein eigenes Gerät (Device) dar. Als Komponenten enthält dieses die einzelnen Messgeräte, welche jeweils durch den MTConnect Komponententyp Sensor als einzelne Sensoreinheit repräsentiert werden [17]. Die gewählten Datenpunkte stellen dann die DataItems der jeweiligen Sensorkomponente dar.

Der Adapter selbst benötigt aus der Benutzeroberfläche für die Konfiguration eigentlich nur den Port, der verwendet werden soll, und eine Liste der verwendeten Messgeräte. Das gewählte Intervall wird nicht für den Adapter verwendet, sondern

als Abfrageintervall für die Messgeräte. Die Messgeräte erhalten als Konfigurationsdaten jeweils die IP des Messgerätes im Netzwerk, einen Namen, eine eindeutige ID und eine Reihe von Datenpunkten, die aus dem Gerät ausgelesen werden sollen. Der Port ist bei den Messgeräten mit ModbusTCP immer gleich 502.

Nachdem der/die AnwenderIn den Adapter über die Benutzeroberfläche konfiguriert hat, kann er/sie den Adapter über die Benutzeroberfläche starten. Das Sequenzdiagramm in Abbildung 20 zeigt den Ablauf des darauffolgenden Prozesses, der durch das Event *AdapterStartButton_Click* ausgelöst wird. Beim ersten Betätigen wird dabei die Methode *StartPacAdapter* aufgerufen, mit welcher zuerst für jedes darin enthaltene Messgerät Pac die *StartPac* Methode aufgerufen wird, bevor der Adapter selbst gestartet wird. Mit diesen Startmethoden werden zuerst die jeweiligen Cycle Klassen erstellt, welche danach in jeweils einem asynchronen Thread gestartet werden, der so lange aktiv ist, bis der jeweilige Boolesche Ausdruck *stopRequested* von *false* auf *true* gesetzt wird. Während der Ausführung der Threads liest jeder einzelne Messgeräte-Thread die jeweils konfigurierten Register aus dem Gerät aus und speichert diese in einem Array. Der *PacAdapterCycle* greift dazu asynchron auf alle Arrays zu und schickt die Werte, die sich verändert haben, an die verbundenen Agents.

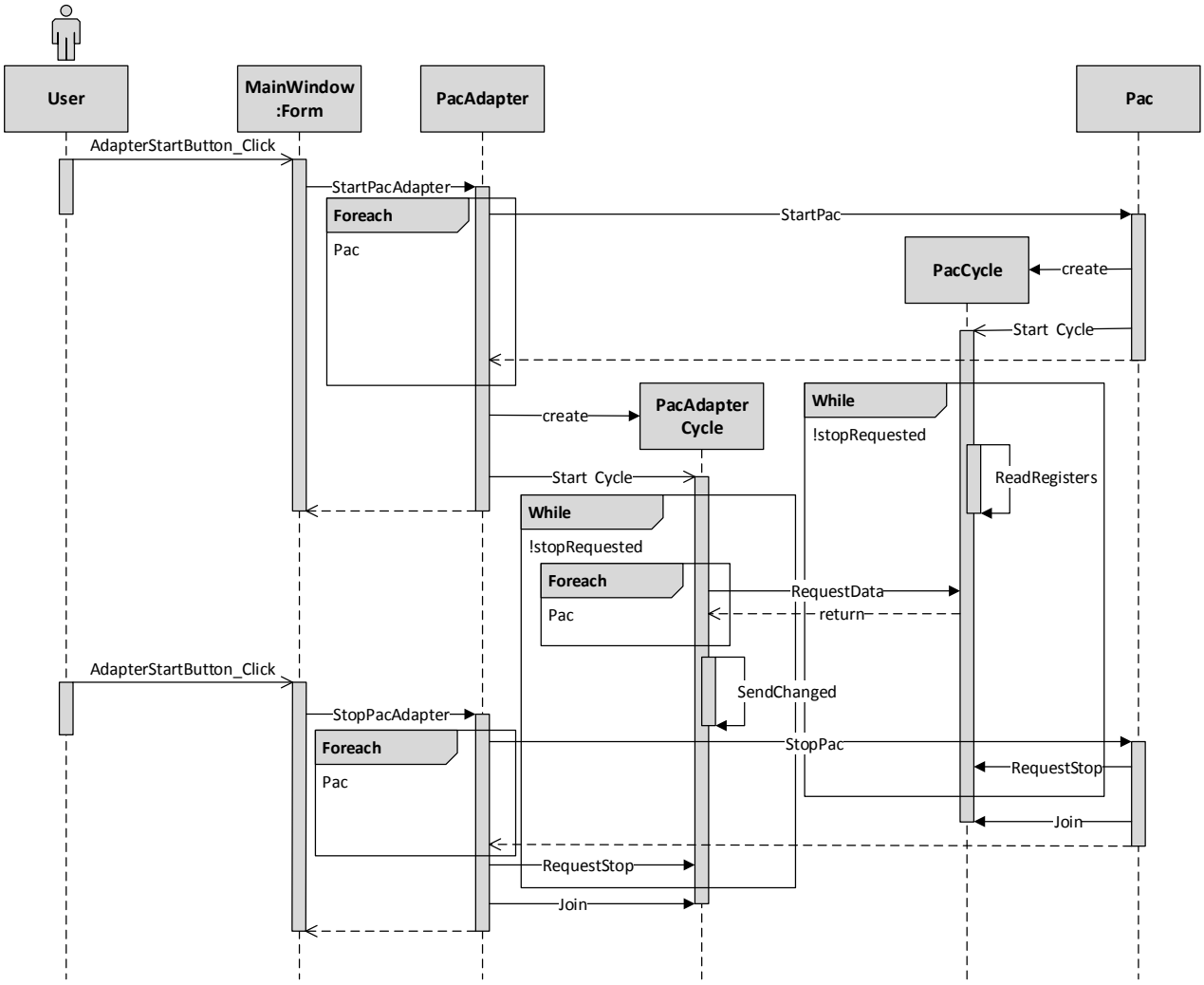

Abbildung 20: Sequenzdiagramm Adapterstart

Eine vereinfachte Darstellung des Ablaufes bietet Abbildung 21. Hier sind die Threads als einzelne Zyklen dargestellt, die unabhängig voneinander arbeiten und dadurch bei Ausfall von einzelnen Messgeräten oder Problemen bei der Kommunikation keine Auswirkung auf die anderen Zyklen oder den Adapterzyklus haben.

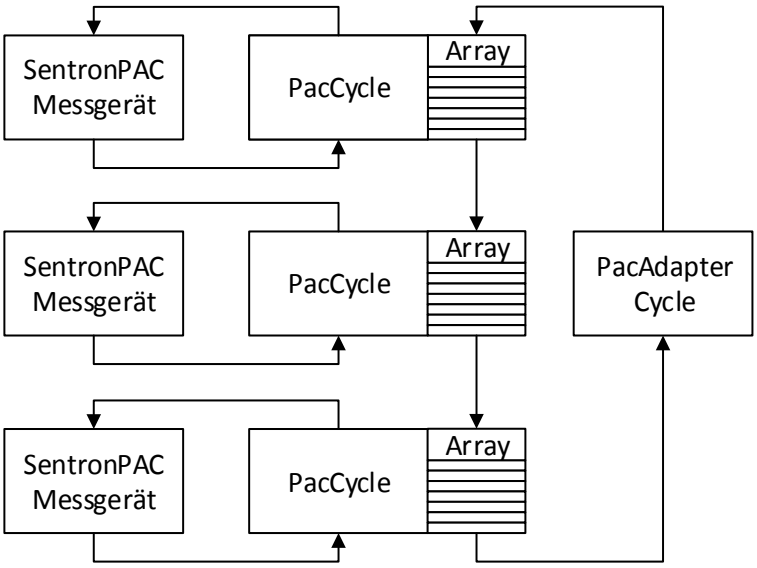

Abbildung 21: Threading für Adapter und Messgeräte

In den einzelnen Abfragezyklen der Messgeräte wird die Methode *ReadRegisters* aufgerufen, mit der der Stringarray *_data* (Code-Snippet 12, Zeile 8), also ein Liste aus Zeichenfolgen, mit den Daten eines Messgerätes beschrieben wird und neben den gelesenen Werten noch zusätzliche Informationen, wie den Identifier des DataItems und die Position des gelesenen Registers enthält. Mit der Funktion *ReadHoldingFloat* (Zeile 7) wird über ModbusTCP eine Anfrage mit dem Funktionscode 3 an das Messgerät geschickt. Mit diesem Code werden zwei Register (jeweils 16 Bit) ab einer bestimmten Startposition ausgelesen und formatiert, so dass beide zusammen als eine Gleitkommazahl mit dem Datentyp Float (32 Bit) zurückgegeben werden können. Wenn die Kommunikation fehlschlägt wird, entsprechend dem MTConnect Standard, der Wert *UNAVAILABLE* an die Stelle im Array geschrieben (Zeile 10).

```
1   private void ReadRegisters()
2   {
3       for (int i = 4; i < _data.Length; i += 3)
4       {
5           try
6           {
7               float value = _sPac.ReadHoldingFloat(_offset[i]);
8               _data[i + 1] = value.ToString();
9           }
10          catch { _data[i + 1] = "UNAVAILABLE"; }
11      }
12  }
```

Code-Snippet 12: ReadRegisters Methode

Für die Übertragung und Verarbeitung (Code-Snippet 12, Zeile 7) der einzelnen Anfragen benötigt ein Thread in der Testumgebung pro Datenpunkt eine Zeit von drei bis vier Millisekunden. Da die Anzahl der gewünschten Datenpunkte und die Dauer pro Anfrage variieren kann, muss die Wartezeit zwischen den Abfragen dem

eingestellten Intervall angepasst werden. Durch eine Zeitmessung am Anfang der Schleife (Code-Snippet 13, Zeile 8) der Messgerätethreads kann am Ende die nötige Zeit errechnet werden, die der Thread warten muss, bis ein neuer Zyklus gestartet werden kann. So können die Intervalle zeitlich optimiert werden.

```
1  public void Cycle()
2  {
3      _sw = new System.Diagnostics.Stopwatch();
4      Connect();
5
6      while (!_stopRequested)
7      {
8          _sw.Start();
9          while (!_stopRequested && !_sPac.Connected)
10         {
11             SetAllUnavailable();
12             try
13             {
14                 _sPac.Connect();
15                 _data[2] = "AVAILABLE";
16             }
17             catch{ Thread.Sleep(5000); }
18         }
19
20         ReadRegisters();
21
22         _sw.Stop();
23         int timeleft = _interval - (int)_sw.ElapsedMilliseconds;
24         if (timeleft <= 0) timeleft = 1;
25         Thread.Sleep(timeleft);
26         _sw.Reset();
27     }
28     _data[2] = "UNAVAILABLE";
29     _sPac.Disconnect();
30 }
```

Code-Snippet 13: PacCycle Zyklus

Außerdem muss bei einem Verbindungsabbruch sichergestellt werden, dass das Messgerät aufs Neue verbunden werden kann (Zeilen 9 bis 18). In diesem Fall wird so lange eine Schleife durchlaufen, bis das Gerät wieder verbunden ist.

Neben den Threads der Messgeräte spielt der Thread für den MTConnect Adapter (Code-Snippet 14) eine wichtige Rolle. Dieser muss nicht nur die Daten der einzelnen Messgeräte zuordnen und an die verbundenen Agents versenden, sondern er muss auch um ein Vielfaches schneller sein, um die Messwerte, die sich verändert haben, zu erfassen, ohne dass dabei ein Wert verloren geht. Diese Aufgabe wird in der Schleife (Zeile 7 bis 21) erledigt. Hierin wird für jedes Messgerät die *RequestData* Methode aufgerufen, mit der der jeweilige Datenarray zurückgegeben wird, welcher dann dem zugehörigen Samplewert des Adapters zugeordnet wird (Zeile 16). Hierbei wird direkt beim Setzen des *Value* Attributes überprüft, ob sich der neue Wert vom vorherigen unterscheidet. Die Klasse Adapter mit dem Objekt *_adapter* stammt aus der MTConnect DotNetAdapterSDK [24] (Kapitel 3.4.8).

```
1   public void Cycle()
2   {
3       string value;
4       bool running = false;
5       _adapter.Start();
6
7       while (!_stopRequested)
8       {
9           int j = 0;
10          foreach (var element in _pacs)
11          {
12              string[] data = element.PacCycle.RequestData();
13              _events[j].Value = data[2];
14              for (int i = 4; i < data.Length; i += 3)
15              {
16                  _pacSamples[j].Samples[i + 1].Value = data[i + 1];
17              }
18              j++;
19          }
20          _adapter.SendChanged();
21      }
22      _adapter.Unavailable();
23      _adapter.SendChanged();
24      _adapter.Stop();
25  }
```

Code-Snippet 14: PacAdapterCycle Zyklus

Wenn alle Arrays verarbeitet wurden, wird die Methode *SendChanged* des Adapters aufgerufen (Zeile 22). Mit dem Aufruf dieser Methode werden alle DataItems des Adapters, deren Werte sich im Vergleich zum vorherigen Zyklus geändert haben, an die verbundenen Agents weitergeleitet. Hierbei wird zuerst der Zeitstempel zum aktuellen Zeitpunkt erstellt und dann eine Zeichenfolge aus dem Zeitstempel, sowie den veränderten DataItems und deren Werten erstellt:

$$TimeStamp|Identifier_1|Value_1|Identifier_2|Value_2 + \ldots + |Identifier_n|Value_n$$

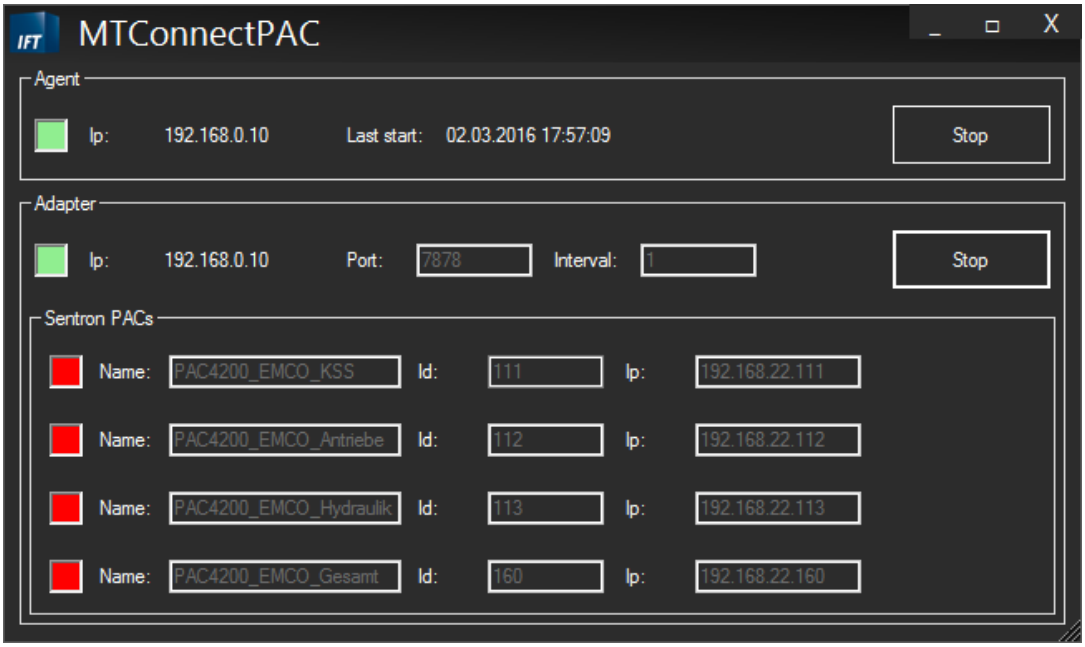

Abbildung 22: Adapter mit laufenden Threads, aber nicht verbundenen Geräten

3.4.4 Agent

Die Klasse Agent dient der Einbindung des MTConnect Agents und beinhaltet Funktionen, um die Konsolenanwendung im Hintergrund zu starten und wieder zu beenden. Außerdem wird der Agent erstellt und die Konfigurationsdateien werden verknüpft, eingelesen und überschrieben. Bei der Erstellung eines Objektes der Klasse Agent wird dabei zuerst geprüft, ob der Agent bereits als Programm im Application Data Ordner vorhanden ist, ansonsten wird er automatisch erstellt. Dazu ist der Agent direkt im Installationsprogramm als komprimierte Datei angehängt und kann bei Bedarf an gezielter Stelle entkomprimiert werden.

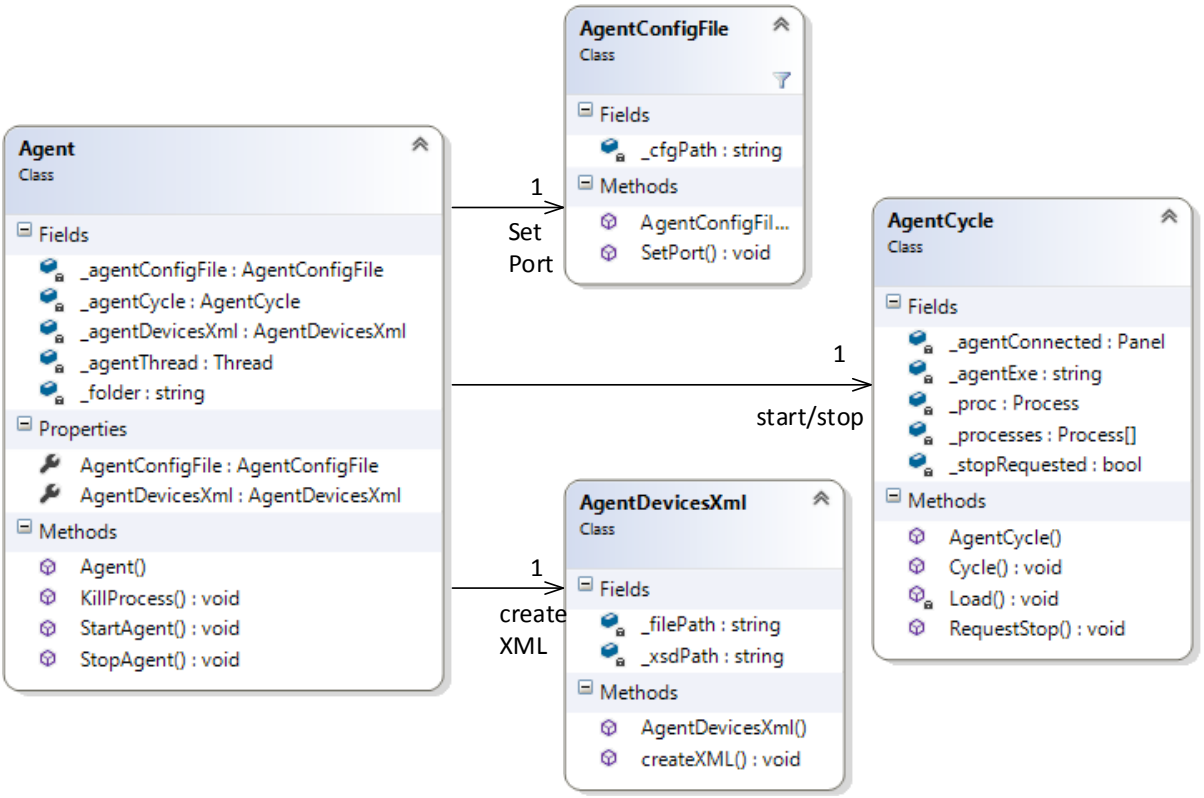

Abbildung 23: Agentklasse und zugehörige Subklassen

Der Thread AgentCycle (Abbildung 23) prüft hierbei nur, ob das Agentprogramm noch problemlos im Hintergrund läuft und startet bzw. beendet außerdem die Konsolenanwendung. Mit der Klasse AgentConfigFile wird die Konfigurationsdatei eingerichtet. Dazu wird diese vor dem Start des Agents eingelesen und der Adapter zu den eventuell schon vorhandenen anderen Adaptern hinzugefügt. Außerdem wird der vom/von der AnwenderIn gewählte Port eingestellt. Die Klasse *AgentDevicesXml* konfiguriert die Datei Devices.xml des Agenten, die die Gerätestruktur nach Teil 2 des MTConnect Schemas beschreibt. Dazu ist es ebenso notwendig die bereits vorhandene Struktur einzulesen, damit die bereits konfigurierte Struktur anderer Geräte erhalten bleibt und nur die Struktur des Adapters neu generiert wird. Diese Klasse nutzt dazu die MTConnectXML Bibliothek (Kapitel 3.4.6), mit der die XML Struktur in Form einer objektorientierten Klassenstruktur erstellt werden kann.

3.4.5 MTConnectPacFile

Die Klasse MTConnectPacFile dient der Speicherung und dem erneuten Laden einer bestimmten Konfiguration durch den/die AnwenderIn. Durch eine einheitliche Struktur im XML Format kann die Konfiguration damit leicht über die Anwendung selbst, durch manuelle Konfiguration oder über andere Anwendungen erfolgen. Die Datei enthält für jedes Messgerät ein Element namens Sensor und eine Reihe von DataItem Elementen. Die ConfigItem Elemente dienen u. a. zur Einstellung von Intervall und Port des Adapters und werden, wenn sie nicht mit angegeben werden, mit ihren default Werten geladen.

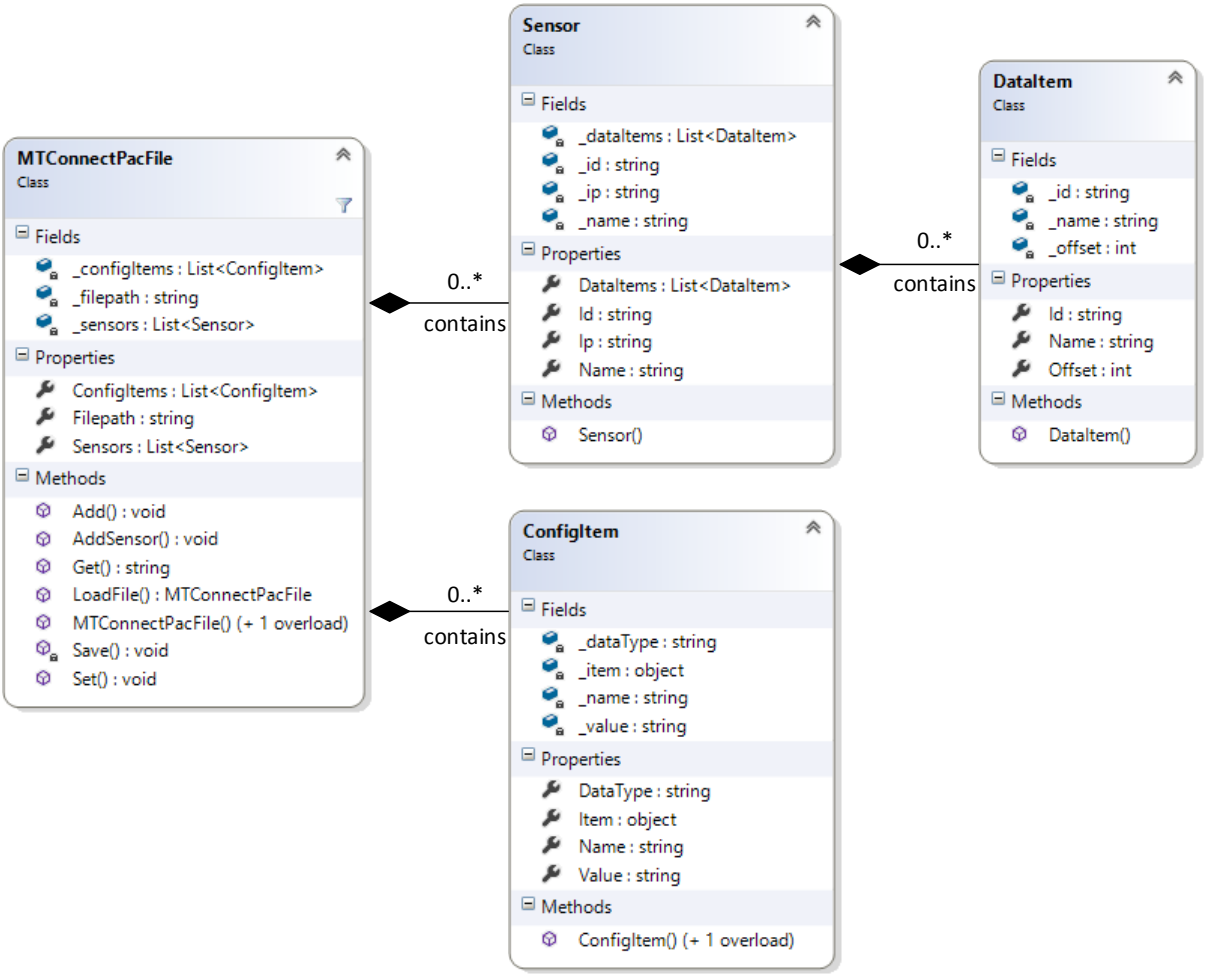

Abbildung 24: Klasse MTConnectPacFile und zugehörige Subklassen

Die auf dieser Klassenstruktur (Abbildung 24) basierende XML Struktur der Konfigurationsdatei ist in Code-Snippet 15 dargestellt und enthält einen Container mit dem Namen Components für die Sensorelemente und einen Container ConfigItems. Die Sensoren benötigen als Attribute einen Namen, eine ID und eine IP. Die zum Sensor gehörigen DataItems benötigen einen Namen und die Registerposition des auszulesenden Wertes im Messgerät. Diese Konfigurationsdatei mit dem Dateiformat .mtp kann vom Programm aus oder direkt per Doppelklick geöffnet werden.

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <MTConnectPacFile xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xmlns:xsd="http://www.w3.org/2001/XMLSchema" Filepath="C:\EMCO.mtp">
3.    <Components>
4.      <Sensor Name="PAC4200_EMCO_KSS" Id="111" Ip="192.168.22.111">
5.        <DataItems>
6.          <DataItem Name="Wirkleistung KSS" Id="111_65" Offset="65" />
7.        </DataItems>
8.      </Sensor>
9.      <Sensor Name="PAC4200_EMCO_Antriebe" Id="112" Ip="192.168.22.112">
10.       <DataItems>
11.         <DataItem Name="Wirkleistung Antriebe" Id="112_65" Offset="65" />
12.       </DataItems>
13.     </Sensor>
14.     <Sensor Name="PAC4200_EMCO_Hydraulik" Id="113" Ip="192.168.22.113">
15.       <DataItems>
16.         <DataItem Name="Wirkleistung Hydraulik" Id="113_65" Offset="65" />
17.       </DataItems>
18.     </Sensor>
19.     <Sensor Name="PAC4200_EMCO_Gesamt" Id="160" Ip="192.168.22.160">
20.       <DataItems>
21.         <DataItem Name="Wirkleistung Gesamt" Id="160_65" Offset="65" />
22.       </DataItems>
23.     </Sensor>
24.   </Components>
25.   <ConfigItems>
26.     <ConfigItem Name="adapterPort" Value="7878" DataType="System.Int32">
27.       <Item xsi:type="xsd:int">7878</Item>
28.     </ConfigItem>
29.     <ConfigItem Name="interval" Value="1" DataType="System.Int32">
30.       <Item xsi:type="xsd:int">1</Item>
31.     </ConfigItem>
32.     <ConfigItem Name="height" Value="384" DataType="System.Int32">
33.       <Item xsi:type="xsd:int">384</Item>
34.     </ConfigItem>
35.     <ConfigItem Name="width" Value="650" DataType="System.Int32">
36.       <Item xsi:type="xsd:int">650</Item>
37.     </ConfigItem>
38.     <ConfigItem Name="max" Value="Normal"
          DataType="System.Windows.Forms.FormWindowState">
39.       <Item xsi:type="xsd:int">0</Item>
40.     </ConfigItem>
41.     <ConfigItem Name="decimalPoint" Value="False" DataType="System.Boolean">
42.       <Item xsi:type="xsd:boolean">false</Item>
43.     </ConfigItem>
44.   </ConfigItems>
45. </MTConnectPacFile>
```

Code-Snippet 15: MTConnectPacFile

Zum Starten des Programmes gibt es daher zwei Methoden. Die eine ist der Programmstart über die normale Ausführungsdatei, die andere der Start über die Konfigurationsdatei. In Abbildung 25 ist der Programmstart als Aktivitätsdiagramm dargestellt. Dabei wird zuerst überprüft, ob nicht schon eine Instanz des Programmes läuft, da sich mehrere aktive Instanzen dieses Programmes auf einem Rechner gegenseitig behindern würden, weil mehrere Adapter nicht gleichzeitig den gleichen Port verwenden können. Wenn das Programm noch nicht aktiv ist, wird eine neue Instanz des Programmes gestartet, bei der geprüft wird, ob der Start von einer Konfigurationsdatei aus erfolgt ist. Wenn nicht, dann wird der StartScreen geladen,

mit dem ein Dateipfad einer vorhandenen Datei oder ein Dateipfad für eine neue Datei gewählt werden kann. Mit dem Pfad kann dann das Hauptfenster (MainScreen) der Anwendung geladen werden (Abbildung 26).

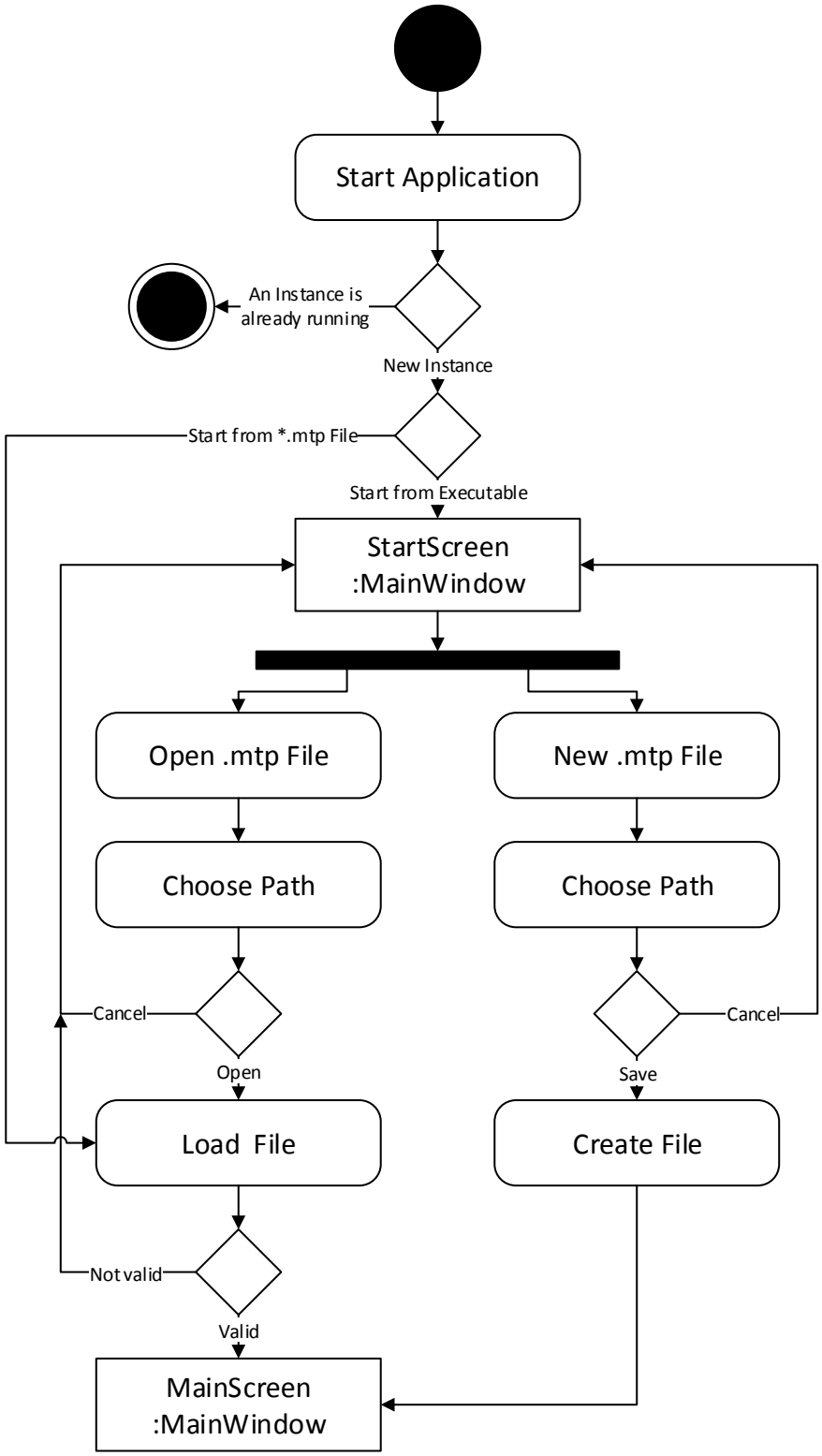

Abbildung 25: Aktivitätsdiagramm Programmstart

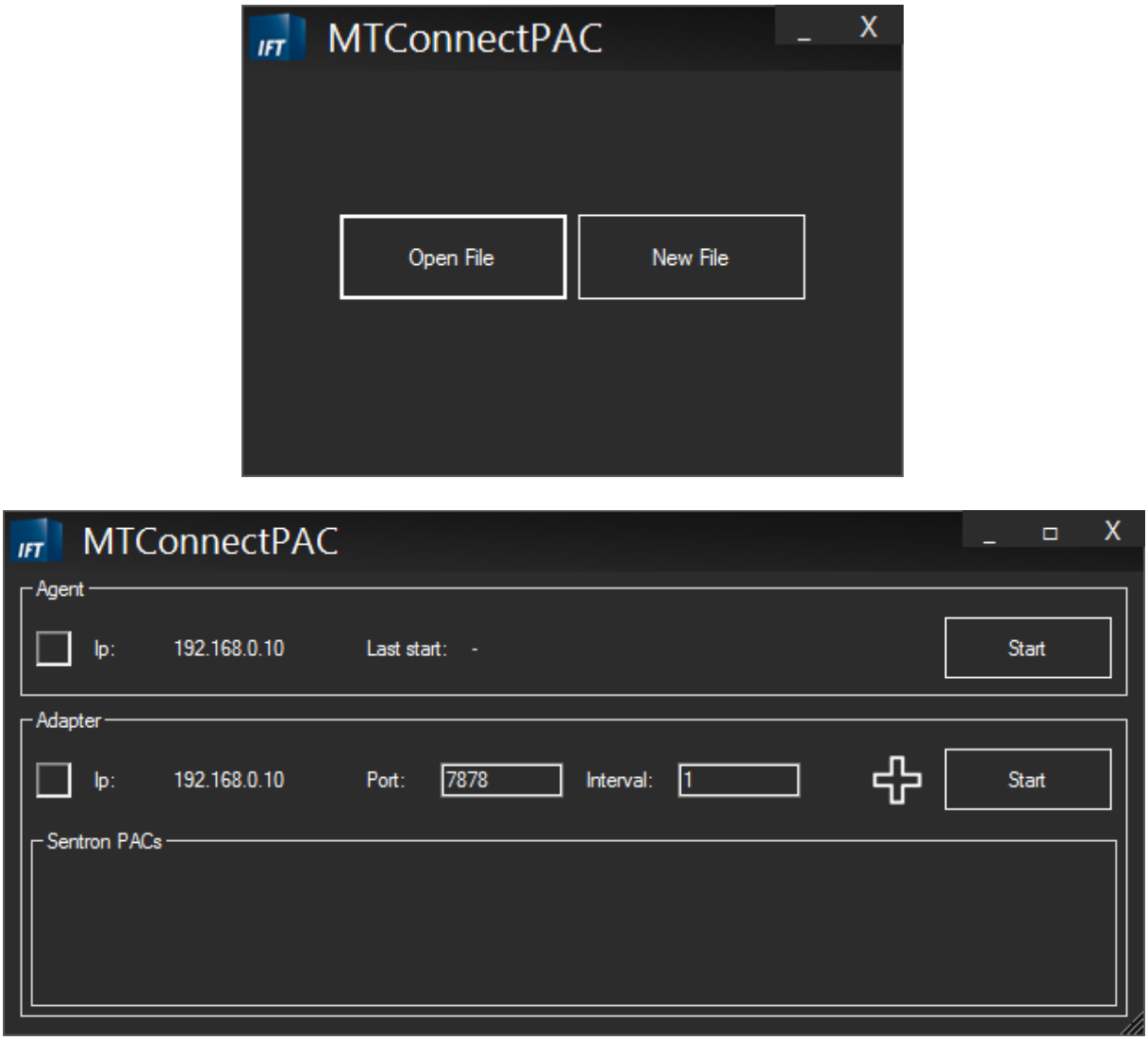

Abbildung 26: StartScreen (oben) und MainScreen (unten)

3.4.6 MTConnectXML Bibliothek

Die MTConnectXML Bibliothek dient der Erstellung der Gerätedatenstruktur für den Agenten, basierend auf einer Klassenstruktur mit Objekten. Hierzu wird eine statische Klassenstruktur verwendet, die durch das XML Schema Definition Tool [28] aus dem MTConnectDevices Schema erstellt wurde. Diese statische Struktur reicht aus, wenn der MTConnect Standard um neue Asset- oder DataItemtypes erweitert werden soll, da Assets über das Assets Schema und die Typen für DataItems im Streamsschema beschrieben werden. Die Datentypen sind in der MTConnect Datenstruktur nach dem MTConnectDevices Schema nur als Attribut und nicht als Element beschrieben und können daher alle möglichen Zeichenfolgen enthalten. Diese statische Struktur würde jedoch nicht ausreichen, wenn ein neuer MTConnect Komponententyp hinzugefügt werden soll, da dieser in der Struktur als Element auftritt und somit in der Klassenstruktur eine eigene Klasse darstellen würde. Daher

erfolgt das Einlesen der restlichen Geräte dynamisch, damit keine Erweiterungen von Componenttypes verloren gehen. Die Gerätestruktur des Adapters wird auf dem statischen Modell basierend erzeugt.

3.4.6.1 Klassenstrukturen aus XML Schemata

Die Erzeugung der statischen Klassenstruktur erfolgt mit dem in Microsoft Visual Studio vorhandenen XSD Tool [28], welches über die Eingabeaufforderung gestartet werden kann. Dadurch wird eine Schemadatei in eine C# Datei mit Klassen umgewandelt. Jedes Element stellt dabei eine Klasse und jedes Attribut eine Eigenschaft dieser Klasse dar.

```
1  [System.CodeDom.Compiler.GeneratedCodeAttribute("xsd", "4.0.30319.33440")]
2  [System.SerializableAttribute()]
3  [System.Diagnostics.DebuggerStepThroughAttribute()]
4  [System.ComponentModel.DesignerCategoryAttribute("code")]
5  [System.Xml.Serialization.XmlTypeAttribute(Namespace =
   "urn:mtconnect.org:MTConnectDevices:1.3")]
6  [System.Xml.Serialization.XmlRootAttribute("MTConnectDevices", Namespace =
   "urn:mtconnect.org:MTConnectDevices:1.3", IsNullable = false)]
7  public partial class MTConnectDevicesType
8  {
9      private HeaderType headerField;
10     private DeviceType[] devicesField;
11
12     public HeaderType Header
13     {
14         get
15         {
16             return this.headerField;
17         }
18         set
19         {
20             this.headerField = value;
21         }
22     }
23
24     [System.Xml.Serialization.XmlArrayItemAttribute("Device", IsNullable = false)]
25     public DeviceType[] Devices
26     {
27         get
28         {
29             return this.devicesField;
30         }
31         set
32         {
33             this.devicesField = value;
34         }
35     }
36 }
```

Code-Snippet 16: MTConnectDevices Element als Klasse

Das oberste Element der Struktur *MTConnectDevices* besteht z. B. aus einem *Header*element und dem Element *Devices*, das als Container für Elemente des Typs *Device* dient. Die entsprechende Klassenstruktur für das MTConnectDevices Element ist in Code-Snippet 16 dargestellt. Dabei enthält die Klasse MTConnectDevices ein Objekt der Klasse Header und einen Array für Objekte der

Klasse Device. Da das Containerelement Devices keine Attribute besitzt, sondern nur eine Reihe an Unterelementen, wird es in der Klassenstruktur nicht mehr benötigt. Für die darunter liegenden Elemente findet die gleiche Klassenumwandlung statt.

3.4.6.2 XML Erzeugung

Durch diese Konvertierung können nun Objekte aus den jeweiligen Klassen erstellt und entsprechend dem Schema miteinander verknüpft werden. Diese Strukturen können eingelesen, geschrieben und an bestimmten Stellen erweitert werden.

```
1   public void createXML(List<Pac> pacs, string filepath)
2   {
3       List<XElement> savedDevices = SaveDevicesFrom(filePath);
4
5       MTConnectDevicesType newAgentConfig = new MTConnectDevicesType();
6
7       DeviceType device = new DeviceType { name = "MTConnect_PAC", id =
            "MTConnect_PAC", uuid = "MTConnect_PAC" };
8       MTConnectXMLTools.AddDeviceToMTConnectDevicesType(device, newAgentConfig);
9
10      foreach (var pac in pacs)
11      {
12          SensorType sentron = new SensorType { name = pac.Name, id = pac.Id };
13          MTConnectXMLTools.AddComponentToDevice(sentron, device);
14
15          foreach (var current in pac.DataConfiguration.SentronPacDataItems)
16          {
17              DataItemType newItem = new DataItemType { name = current.Name, id =
                    pac.Id + "_" + current.Offset, type = current.Type, units =
                    current.Unit, category = CategoryType.SAMPLE };
18              MTConnectXMLTools.AddDataItemToObject(newItem, sentron);
19          }
20      }
21
22      MTConnectXMLTools.writeToXML(newAgentConfig, savedDevices, filePath);
23  }
```

Code-Snippet 17: Programmatische Erstellung der XML Konfiguration

In Code-Snippet 17 ist die Erstellung der XML Struktur über Klassen dargestellt. Dafür werden zuerst die aktuellen XML Strukturen der anderen Geräte dynamisch und vollständig in die Liste savedDevices eingelesen (Zeile 3), damit die Strukturen dieser Geräte erhalten bleiben. Danach wird ein Objekt für die neue Konfiguration erstellt (Zeile 5) und das Device MTConnect_PAC hinzugefügt (Zeilen 7 bis 8). Diesem Device wird dann für jedes konfigurierte Messgerät ein Sensorelement (Zeilen 12 bis 13) und diesem Sensorelement die jeweiligen Datenpunkte (Zeile 17 bis 18) hinzugefügt. Zuletzt wird die Klassenstruktur in eine XML Struktur umgewandelt (Zeile 22), der dann die anfangs im XML Format gesicherten restlichen Geräte wieder hinzugefügt werden können.

3.4.7 SentronPAC Bibliothek

Die SentronPAC Bibliothek dient der Kommunikation mit den SentronPAC Messgeräten und nutzt dafür die EasyModbus Bibliothek [25]. Sie nutzt die Daten einer XML Tabelle, die vom/von der NutzerIn angepasst und um weitere Zeilen erweitert werden kann. Sie enthält Daten entsprechend der technischen Dokumentation der Messgeräte [29], erweitert um deren MTConnect spezifische Daten wie Kategorie, Einheit und DataItemtype.

```
<Table1>
  <F1>127</F1>
  <F2>2</F2>
  <F3>-</F3>
  <F4>Maximum</F4>
  <F5>THD-R</F5>
  <F6>Current</F6>
  <F7>c</F7>
  <F8>Float</F8>
  <F9>PERCENT</F9>
  <F10>0</F10>
  <F11>to</F11>
  <F12>100</F12>
  <F13>ReadOnly</F13>
  <F14>Sample</F14>
  <F15>e:TOTAL_HARMONIC_DISTORTION</F15>
  <F16>-</F16>
</Table1>
```

Code-Snippet 18: Eine Zeile in der XML Tabelle mit 16 Spalten

Code-Snippet 18 stellt eine einzelne Zeile der Tabelle für den Messwert *Maximum THD-R Current C* dar. Die Spalten enthalten Informationen zu Offset, Anzahl der gelesenen Register, Namen, Datentyp, Einheit, Messbereich, Lesbarkeit, Kategorie und DataItemtype. Basierend auf dieser Tabelle wird im Adapterprogramm die Liste der auslesbaren Datenpunkte (Abbildung 18) erstellt.

3.4.8 MTConnect DotNetAdapterSDK

Die MTConnect DotNetSDK [24] ist eine freie Bibliothek zur Implementation eines MTConnect Adapters (Abbildung 27). Sie umfasst die verschiedenen Arten von *DataItem* Kategorien und das Asset *CuttingTool*.

Der Adapter enthält einen Thread *ListenForClients*, mit dem Verbindungen von Agents angenommen und für jeden verbundenen Agent in einem weiteren Thread *HeartbeatClient* aufrechterhalten werden. Hierüber werden die Heartbeat Signale, die der Agent in bestimmten Abständen sendet, beantwortet. Dabei sendet ein Agent die Zeichenfolge *PING* an den konfigurierten Adapter und der Adapter antwortet daraufhin mit der Zeichenfolge *PONG*. Über diese Funktion hält der Adapter eine Liste mit verbundenen Agents aufrecht. Der Thread *ListenForClients* wird über die

Funktionen *Start* und *Stop* gestartet und wieder beendet. Dadurch werden bereits alle passiven Aufgaben eines Adapters abgedeckt.

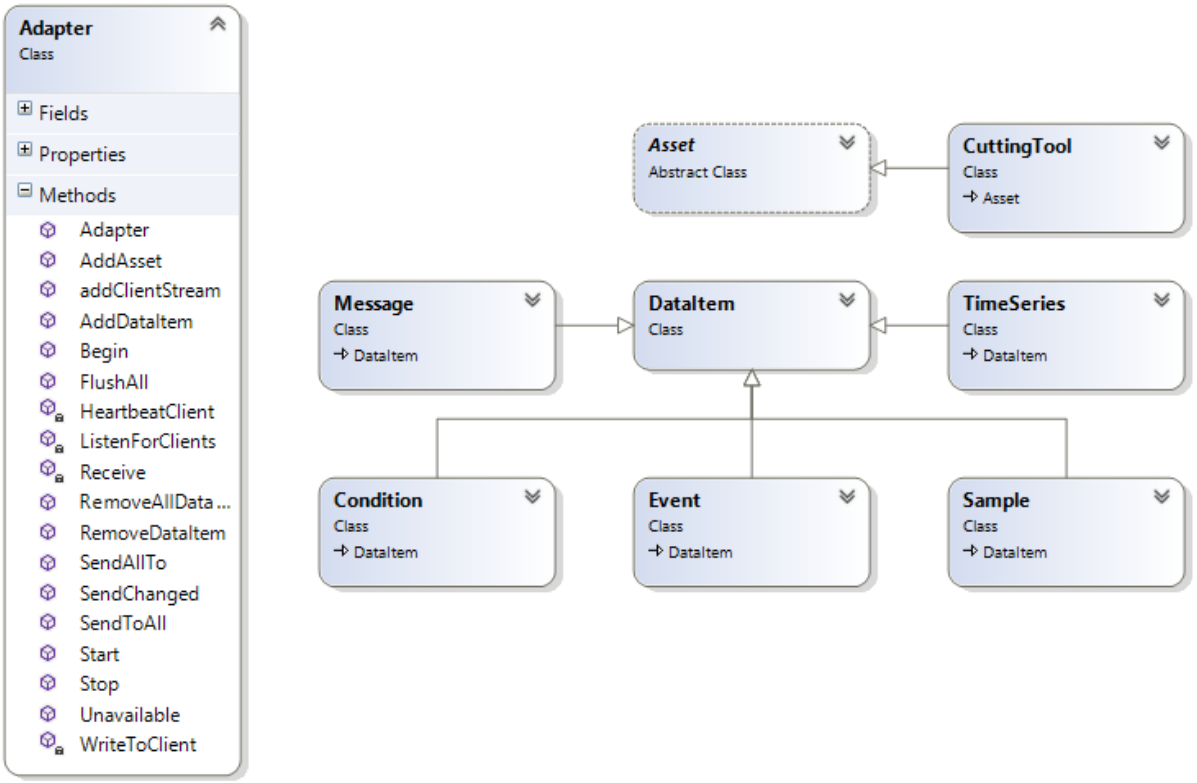

Abbildung 27: MTConnect Dot Net SDK – Klassendiagramm

Neben diesen Aufgaben muss der Adapter auch DataItems mit den zugehörigen Werten und Zeitstempeln an die verbundenen Agents senden. Dazu werden dem Adapter über die Funktion *AddDataItem* Objekte der von *DataItem* abgeleiteten Klassen hinzugefügt. Den einzelnen DataItems können dann kontinuierlich Werte zugeordnet werden, welche vom Adapter auf eine Veränderung gegenüber dem vorherigen Wert geprüft werden. Durch die Methode *SendChanged* werden die Werte der veränderten DataItems zu Zeichenfolgen entsprechend dem MTConnect Adapter Protokoll verbunden und an die Agents gesendet.

3.5 Adapter des Drehzentrums

Auf der Drehmaschine soll direkt auf der Steuerung ein MTConnect Adapter als Fertigprodukt von einem Lieferanten installiert werden, der durch den unmittelbaren Datenzugriff alle notwendigen Parameter erfassen kann. Die Integration des Informationsmodells der Drehmaschine erfolgt nach der Struktur in Abbildung 28 in dem Agent, der auch das Informationsmodell der Messgeräte besitzt.

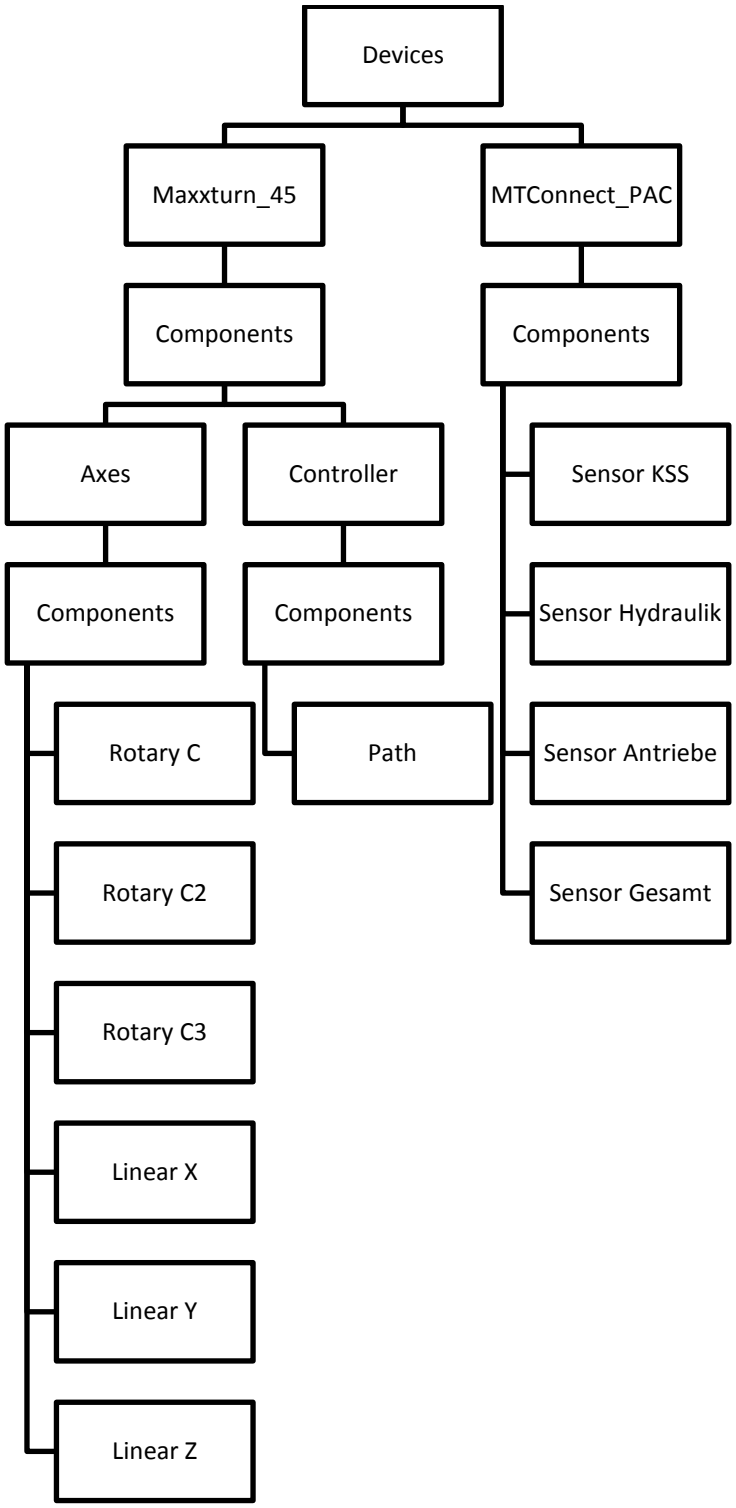

Abbildung 28: Gesamtes Informationsmodell

Basierend auf den Gerätespezifikationen der Drehmaschine enthält das Informationsmodell die linearen Achsen, die Längs- (Z) und Querachsen (X und Y), die Rotationsachsen für die Haupt- (C) und Gegenspindel (C2) und eine Rotationsachse für den Antrieb des Werkzeugrevolvers (C3). Außerdem besitzt die Maschine die Komponente *Controller*, welche die Numerical Control (NC) der Maschine repräsentiert und eine Subkomponente Path besitzt, von der die programmspezifischen Operationen eines NC Kanals durchgeführt werden.

	Type	SubType	Category	Units
Linear X, Y, Z	POSITION	ACTUAL	SAMPLE	MILLIMETER
	LOAD		SAMPLE	PERCENT
Rotary C, C2, C3	ROTARY_VELOCITY	ACTUAL	SAMPLE	REVOLUTION /MINUTE
	LOAD		SAMPLE	PERCENT
	ROTARY_MODE		EVENT	
	ROTARY_VELOCITY_OVERRIDE		EVENT	PERCENT
	DIRECTION	ROTARY	EVENT	
Controller	MESSAGE		EVENT	
	EMERGENCY_STOP		EVENT	
Path	CONTROLLER_MODE		EVENT	
	EXECUTION		EVENT	
	PROGRAM		EVENT	
	BLOCK		EVENT	
	LINE		EVENT	
	PROGRAM_COMMENT		EVENT	
	TOOL_ASSET_ID		EVENT	
	PATH_FEEDRATE	ACTUAL	SAMPLE	MILLIMETER /SECOND
	PATH_FEEDRATE	COMMANDED	SAMPLE	MILLIMETER /SECOND
	PATH_FEEDRATE_OVERRIDE	JOG	EVENT	PERCENT
	PATH_FEEDRATE_OVERRIDE	RAPID	EVENT	PERCENT
	PATH_POSITION	ACTUAL	SAMPLE	MILLIMETER_3D
	PART_COUNT	ALL	EVENT	

Tabelle 2: Event und Sample DataItems der Drehmaschine

Tabelle 2 zeigt die den jeweiligen Komponenten zugeordneten DataItems entsprechend den Angaben der Adaptersoftware und der Version 1.3.1 des MTConnect Standards. So liefern die linearen Achsen hauptsächlich aktuelle Positionsdaten in Maschinenkoordinaten und die rotierenden Achsen Drehzahl und Drehrichtung. Der NC Kanal liefert dazu den Programmnamen, den aktuellen Block im NC Programm, sowie die zugehörige Zeilennummer. *EXECUTION* zeigt dabei an, ob die Maschine aktiv ist und *TOOL_ASSET_ID* liefert die der Maschine bekannte ID des aktuell eingewechselten Werkzeugs. Außerdem wird die aktuelle Vorschubgeschwindigkeit angegeben.

	Type	SubType	Category	Units
Sensor	AVAILABILITY	ACTUAL	EVENT	
	WATTAGE		SAMPLE	WATT

Tabelle 3: Event und Sample DataItems der einzelnen Messgeräte

Dazu erhält der Agent die entsprechenden elektrischen Leistungswerte *WATTAGE* der Sentron PAC Messgeräte für Hydraulik-, Kühlschmierstoff- (KSS), Antriebs- und Gesamtsystem (Tabelle 3).

4 Zusammenfassung und Ausblick

Letztendlich wurde eine Integration der Maschinendaten und elektrischen Leistungen einer Drehmaschine basierend auf dem Informationsmodell von MTConnect durchgeführt. Die Maschinendaten der Adaptersoftware auf der Maschine wurden dabei mit den Sensordaten der Messgeräte in ein gemeinsames Informationsmodell in einem Agent zusammengeführt und so für weitere Informationsverarbeitung über das Netzwerk zur Verfügung gestellt. Die Maschine ist damit bereit für weiterführende Untersuchungen, die durch die vereinfachte Zugänglichkeit von Daten durch den MTConnect Standard begünstigt werden.

Es hat sich außerdem gezeigt, dass mit MTConnect vor allem eine einfache und kostengünstige Methode geboten wird, die durch freie Software, wie den Agent, und freie Softwarebibliotheken in vielen Programmiersprachen begünstigt wird. Da die Kommunikation über Sockets ein sehr einfaches und grundlegendes Kommunikationsprotokoll darstellt, sollte die native Einbindung des MTConnect Standards, besonders für Hersteller von Werkzeugmaschinen oder Steuerungen, eine schnell umsetzbare und günstige Methode sein, um ihre Maschinen für zukünftige Anforderungen zu rüsten. MTConnect bietet dabei nicht nur für die großen Unternehmen, sondern besonders auch für kleine und mittlere Unternehmen (KMU) eine kostengünstige Schnittstelle, um die Vernetzung ihres Shopfloors voranzutreiben.

4.1 Zukünftige Entwicklung

Zukünftig wird der MTConnect Standard immer umfangreicher werden und nicht nur die standardisierte Integration von Schneidwerkzeugen als Assets beinhalten, sondern auch die Verfolgung von Werkstücken (*Part*) und deren Qualitätskontrollen (Inspection) erlauben [30]. Dies wird eine noch bessere Einbindung von Konzepten wie *Total Productive Management* (TPM) und *Overall Equipment Effectiveness* (OEE) erlauben, wenn die Bearbeitungszeiten und die Prozessschritte für einzelne Werkstücke nachverfolgt und anschließend optimiert werden können. In diesem Bereich gab es bereits erste Ansätze, in denen der Standard *Quality Measurement Results* (QMResults), auf Basis einer automatisierten Qualitätsmessung von Werkstücken, als Erweiterung des Assetsschemas von MTConnect integriert wurde [31]. Besonders moderne Instandhaltungsansätze, wie E-Maintenance, benötigen eine umfangreiche Datenüberwachung, um prädiktive Informationen - eventuell auf Basis künstlicher Intelligenz - zu liefern [32]. Demnach könnte MTConnect als nicht proprietärer Standard als Enabler für solche Instandhaltungskonzepte gesehen

werden, die auf eine solche Überwachung angewiesen sind. Das *Part* Asset würde auch die Integration eines *Smart Products*, welches während der Produktion produktspezifische und individuelle Daten sammelt [33], erlauben.

Eine Erweiterung des MTConnect Standards um zusätzliche Assets bietet ebenso vielversprechende Aussichten. Denkbar wäre es Werkzeugmaschinen auf diese Weise nicht nur Werkzeugdaten, sondern auch NC-Programme, Produktinformationen, Fertigungsaufträge und Materialdaten zentral auf einem oder mehreren Agents bereitzustellen. Werkzeugmaschinen könnten bestimmte Werkstücke, unter Einbindung entsprechender Technologien, wie RFID, erkennen, und von einem Agent die dafür spezifischen Asset-Daten auslesen. Das Asset könnte wiederrum auf ein NC-Programm-Asset referenzieren, welches daraufhin von der Werkzeugmaschine verwendet wird. Über eine sichere Verbindung könnte der Zugriff auf den Agent für Fabriken auf der ganzen Welt in Sekundenschnelle ermöglicht werden. Werkzeughersteller könnten zudem die Archetypen ihrer Werkzeuge auf MTConnect Agents öffentlich bereitstellen, so dass neue Werkzeuge beim Einlesen durch Maschinen automatisch auf ihren Archetyp referenziert werden können und dessen Struktur übernommen werden kann.

Ebenso fehlt dem MTConnect Standard noch die Repräsentation der Geometriedaten der Werkstücke, wie sie beispielsweise im STEP NC (Standard for the Exchange of Product Model Data) Standard verwendet werden [34]. Shin et al. erkannten, dass für die Analyse von Energieverbräuchen von Werkzeugmaschinen auch die Geometrie- und Materialdaten der Werkstücke für die Analyse mit einem Machine Learning Ansatz eine Rolle spielten und verwendeten STEP NC retrospektiv für die Zuweisung [6]. Demnach könnten STEP NC Daten ebenso dem MTConnect Standard als werkstückspezifisches Asset hinzugefügt werden, wenn diese in eine standardisierte XML Struktur umgewandelt werden.

4.2 Nachteile und Kritik

Es gibt auch Kritik am Standard. So bemängelt Oborski, dass wegen des TCP/IP Kommunikationsstandards die Echtzeitfähigkeit des Protokolls stark eingeschränkt bzw. nicht gegeben ist [32]. Da MTConnect jedoch in den meisten Anwendungsszenarien nur eine *Read-only* Lösung darstellt, mit der Daten zu Analysezwecken gesammelt werden, ist die Notwendigkeit einer Echtzeitkommunikation nicht gegeben. MTConnect wird nicht zur Steuerung von Maschinenkomponenten verwendet, welche eine Echtzeitkommunikation benötigen würden. In Szenarien, die einen schnellen Datenaustausch benötigen, sollten die *Streaming* Funktionalitäten (siehe *Interval=0* - Tabelle 1) von MTConnect für die gegebenen Aufgaben ausreichend sein. Selbst bei der Integration von Interfaces [21]

in den Standard, mit welchen die Schnittstelle (z. B. DoorInterface) zwischen zwei Maschinen oder Komponenten, ähnlich einer State-Machine, gehandhabt wird, sollte dies ausreichen.

Ein weiterer Kritikpunkt ist, dass bei MTConnect die Sicherheit sehr eingeschränkt ist, da das Protokoll keinerlei Verschlüsselung oder Autorisierungsebenen vorsieht. Jasperneite, Neumann und Pethig kritisieren, dass die Begründung des reinen Lesezugriffs für die Frage der Sicherheit nicht ausreicht und empfehlen zumindest die Verschlüsselung des Datentransports, um die Vertraulichkeit der Daten zu gewährleisten [16]. Das Thema muss besonders dann stärker betrachtet werden, wenn sich die bisher noch recht übersichtliche Schnittstelle der MTConnect Interfaces weiterentwickelt.

4.3 Entwicklung im Rahmen von Industrie 4.0

Schlussendlich stellt sich die Frage, wie sich MTConnect in zukünftigen Problemstellungen und Szenarien behaupten kann, da besonders in der Hinsicht zum Schwerpunktthema I4.0 “bereits Entscheidungen zur Nutzung von OPC UA getroffen“ wurden [1] [16]. Das ist besonders daher nicht verwunderlich, da OPC UA, im Gegensatz zu MTConnect eine bidirektionale Kommunikation mit Methodenaufrufen und *Read-and-Write* Variablen erlaubt. Außerdem unterstützt es eine offenere Modellierung, die keine Erweiterung des Informationsmodells, wie bei MTConnect, benötigt. Die große Stärke, die MTConnect durch das Informationsmodell und der daraus resultierenden Vereinheitlichung von Gerätedatenmodellen besitzt, wird zur Schwäche, wenn komplexere Anlagen modelliert werden sollen. Dennoch hat MTConnect ebenso entscheidende Vorteile, die wiederum auf das Vorhandensein eines Informationsmodells zurückzuführen sind. Dazu zählen Einfachheit und Einheitlichkeit der Modellierung von Werkzeugmaschinen, was für zukünftige Modellierungsansätze von gesamten Fabriken eine uniforme Grundlage schafft. Damit könnte MTConnect auch in näherer Zukunft zumindest den Bereich der Werkzeugmaschinen beherrschen, denn auf weite Sicht wird auch in diesem Bereich eine bidirektionale Kommunikation notwendig sein. Bei dieser ist es fraglich, ob sie über MTConnect Interfaces durchsetzbar ist oder durch einen anderen Kommunikationsstandard übernommen wird und MTConnect so als unkomplizierter, nicht proprietärer Standard zur Datenerfassung erhalten bleibt.

Trotzdem ist MTConnect zum aktuellen Zeitpunkt schon eine einfache Lösung für I4.0 Szenarien. Dabei erfüllt ein Gerät, das allein mit einem MTConnect Adapter ausgestattet ist noch nicht die Anforderungen die nach RAMI4.0 [1] an eine I4.0 Komponente gestellt werden, denn diesem fehlt die Fähigkeit einen

Kommunikationspartner über die virtuelle Repräsentation des Gerätes in einem Datenmodell zu informieren. Durch den MTConnect Agent und die damit verbundene Funktion zur Modellierung des Gerätes würden jedoch Agent und Adapter zusammen eine I4.0 Komponente ausmachen. Der Adapter repräsentiert an sich nur einen *individuell bekannten* Gegenstand. Durch die Modellierung in einem Agent und die Funktion des Agents, dieses Modell weiterzugeben, wird aus dem Gegenstand eine *Entität*. Diese ist eine Instanz der durch das MTConnect Geräteschema beschriebenen Typen und erlaubt Kommunikationspartnern die Daten des Adapters mit dem Informationsmodell zu referenzieren und auf die zugrundeliegenden Typen Rückschlüsse zu ziehen.

Basierend auf diesen Ansätzen könnte die Modellierung einer Fabrik in Zukunft auf höheren Ebenen und zwischen Geräten mit OPC UA erfolgen. Auf Shopfloor Ebene wäre MTConnect als einheitlicher Standard zur Modellierung und Datenerfassung von Werkzeugmaschinen nach dem standardisierten Informationsmodell denkbar. Aufgrund der einfachen Kommunikation und der bereits nach *MTConnect OPC UA Companion Specification* [35] festgelegten Typenumwandlungen, wäre MTConnect auch standardisiert in ein OPC UA Modell integrierbar. Diese Spezifikation legt fest wie einzelne Typendefinitionen nach dem MTConnect Informationsmodell in OPC UA modelliert werden müssen. Betrachtet wurden hierfür Typendefinitionen für Komponenten, Datentypen und Assets, insbesondere dem Asset-Typ CuttingTool. OPC UA könnte außerdem eine standardisierte Einbindung von MTConnect Interfaces als OPC UA Methoden erlauben und so zumindest Systeme mit festgelegten Zuständen steuern.

5 Literaturverzeichnis

[1] VDI Statusreport, „Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0),“ April 2015. [Online]. Available: https://www.vdi.de/fileadmin/user_upload/VDI-GMA_Statusreport_Referenzarchitekturmodell-Industrie40.pdf. [Zugriff am 15 April 2016].

[2] B. Edrington, B. Zhao, A. Hansel, M. Mori und M. Fujishima, „Machine monitoring system based on MTConnect technology,“ *Procedia CIRP 22,* pp. 92-97, 2014.

[3] A. Vijayaraghavan und D. Dornfeld, „Automated energy monitoring of machine tools,“ *CIRP Annals - Manufacturing Technology 59,* pp. 21-24, 2010.

[4] R. Lynn, A. Chen, S. Locks, C. Nath und T. Kurfess, „Intelligent and Accessible Data Flow Architectures for Manufacturing System Optimization,“ *IFIP Advances in Production Management Systems,* pp. 27-35, 7-9 September 2015.

[5] S. Atluru, S. H. Huang und J. P. Snyder, „A smart machine supervisory system framework,“ *The International Journal of Advanced Manufacturing Technology 58,* pp. 563-572, 2012.

[6] S.-J. Shin, J. Woo und S. Rachuri, „Predictive analytics model for power consumption in manufacturing,“ *Procedia CIRP 15,* pp. 153-158, 2014.

[7] R. A. P. Filleti, D. A. L. Silva, E. J. Silva und A. R. Ometto, „Dynamic System for Life Cycle Inventory and Impact Assessment of Manufacturing Processes,“ *Procedia CIRP 15,* pp. 531-536, 2014.

[8] A. Moos, XQuery und SQL/XML in DB2-Datenbanken, Verwaltung und Erzeugung von XML-Dokumenten in DB2, Wiesbaden: Vieweg+Teubner Verlag, 2008.

[9] J. Fawcett, D. Ayers und L. R. E. Quin, Beginning XML, Indianapolis, Ind.: Wrox, 2012.

[10] A. Salminen und F. Tompa, Communicating with XML, New York Dortrecht Heidelberg London: Springer, 2011.

[11] W3C, „XML Schema Part 2: Datatypes Second Edition,“ W3C Recommendation, 28 Oktober 2004. [Online]. Available: http://www.w3.org/TR/xmlschema-2/#built-in-datatypes. [Zugriff am 15 Dezember 2015].

[12] AMT – The Association For Manufacturing Technology, „Getting Started with MTConnect - Connectivity Guide,“ 1 Oktober 2011. [Online]. Available: http://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/55a93d00e4b03e1c028d77fb/1437154560450/Getting+Started+with+MTConnect+-+FINAL.pdf. [Zugriff am 15 Dezember 2015].

[13] MTConnect Institute, „MTConnect Standard,“ Juni 2015. [Online]. Available: http://www.mtconnect.org/standard?terms=on. [Zugriff am 28 Dezember 2015].

[14] W. Sobel, „MTConnect® Standard - Part 4.0 – Assets - Version 1.3.0,“ 30 September 2014. [Online]. Available: http://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/557f290fe4b09b7d71fc6c52/1434396943780/mtc_part_4_assets_v1.3.pdf. [Zugriff am 15 Dezember 2015].

[15] W. Sobel, „MTConnect® Standard - Part 1 - Overview and Protocol - Version 1.3.0,“ 30 September 2014. [Online]. Available: http://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/557f2897e4b04b2acdba80b5/1434396823825/mtc_part_1_overview_v1.3.pdf. [Zugriff am 15 Dezember 2015].

[16] J. Jasperneite, A. Neumann und F. Pethig, „OPC UA versus MTConnect,“ Juli 2015. [Online]. Available: https://www.researchgate.net/publication/278242232. [Zugriff am 17 Februar 2016].

[17] J. Turner, „MTConnect® Standard - Part 2 - Device Information Model - Version 1.3.1,“ 11 Juni 2015. [Online]. Available: http://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/557af7d6e4b070455fec768d/1434122198226/MTC_Part_2_Components++1.3.1+Final.pdf. [Zugriff am 15 Dezember 2015].

[18] J. Turner, „MTConnect® Standard - Part 3 – Streams Information Model - Version 1.3.1,“ 8 Juni 2015. [Online]. Available: http://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/557acdbae4b0d4d1d367b696/1434111418321/MTC_Part_3_Streams_1+3+1_FINAL.pdf. [Zugriff am 15 Dezember 2015].

[19] W. Sobel, „MTConnect C++ Agent Version 1.3.0.0,“ MTConnect Institute, 8 Mai 2015. [Online]. Available: https://github.com/mtconnect/cppagent/blob/master/README.md. [Zugriff am 9 Februar 2016].

[20] W. Sobel, „MTConnect® Standard - Part 4.1 – Cutting Tools - Version 1.3.0,“ 30 September 2014. [Online]. Available: http://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/557f2920e4b025137e0dd1eb/1434396960393/mtc_part_4.1_cutting_tools_v1.3.pdf. [Zugriff am 15 Dezember 2015].

[21] W. Sobel, „MTConnect® Standard - Part 3.1 – Interfaces - Version 1.3.0,“ 30 September 2014. [Online]. Available: http://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/557f28fde4b09b7d71fc6bcf/1434396925113/mtc_part_3.1_interfaces_v1.3.pdf. [Zugriff am 15 Dezember 2015].

[22] W. Sobel, „cppagent,“ MTConnect Institute, 10 März 2016. [Online]. Available: https://github.com/mtconnect/cppagent. [Zugriff am 21 März 2016].

[23] H. Balzert, Java: Objektorientiert programmieren: Vom objektorientierten Analysemodell bis zum objektorientierten Programm, Herdecke Witten: W3L Verlag, 2010.

[24] W. Sobel, „dot_net_sdk,“ MTConnect Institute, 31 Juli 2014. [Online]. Available: https://github.com/mtconnect/dot_net_sdk. [Zugriff am 21 März 2016].

[25] S. Roßmann, „EasymodbusTCP Modbus Library for .NET and Java,“ 18 Dezember 2015. [Online]. Available: http://easymodbustcp.net/licenseinfo. [Zugriff am 21 März 2016].

[26] Microsoft, „Form Class,“ Microsoft, 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/system.windows.forms.form(v=vs.110).aspx. [Zugriff am 30 März 2016].

[27] Microsoft, „Panel Class,“ Microsoft, 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/system.windows.forms.panel(v=vs.110).aspx. [Zugriff am 30 März 2016].

[28] „XML Schema Definition Tool,“ Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/x6c1kb0s(v=vs.100).aspx. [Zugriff am

21 März 2016].

[29] Siemens, Power Monitoring Device SENTRON PAC3200: Manual, Nürnberg: Siemens AG, 2008.

[30] W. Sobel, „MTConnect Architecture and Research Overview,“ 9 Juni 2015. [Online]. Available: http://www.mtconnect.org/s/MTConnectTechnicalWorkshopAtNAMRC43-x6n1.pdf. [Zugriff am 14 April 2016].

[31] J. L. Michaloski, Y. F. Zhao, B. E. Lee und W. G. Rippey, „Web-enabled, Real-time, Quality Assurance for Machining Production Systems,“ *Procedia CIRP 10,* pp. 332-339, 2013.

[32] P. Oborski, „Developments in integration of advanced monitoring systems,“ *The International Journal of Advanced Manufacturing Technology 75,* pp. 1613-1632, 2014.

[33] D. Kolberg und D. Zühlke, „Lean Automation enabled by Industry 4.0 Technologies,“ *IFAC-PapersOnLine 48-3,* pp. 1870-1875, 2015.

[34] L. Changqing, L. Yingguang und S. Weiming, „Integrated manufacturing process planning and control based on intelligent agents and multi-dimension features,“ *The International Journal of Advanced Manufacturing Technology,* pp. 1457-1471, 2014.

[35] MTConnect Institute, „MTConnect OPC UA Companion Specification Release Candidate - Version 1.02,“ 20 November 2013. [Online]. Available: http://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/5581baf6e4b09b87d6d781d5/1434565366350/mtconnect-opc-ua-companion-specification-v12-release-candidate.pdf. [Zugriff am 15 Dezember 2015].

6 Abbildungsverzeichnis

7 Code-Snippet-Verzeichnis

8 Tabellenverzeichnis

9 Abkürzungsverzeichnis

bzw.	beziehungsweise
Cam-X	Computer Aided Manufacturing using XML
CDATA	Character Data
et al.	Et alia
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
I4.0	Industrie 4.0
ID	Identifier
IoT	Internet of Things
IP	Internet Protocol
KMU	Kleine und mittlere Unternehmen
KSS	Kühlschmierstoff
LCA	Life Cycle Assessment
NC	Numerical Control
NCU	Numerical Control Unit
OEE	Overall Equipment Effectiveness
OPC	OLE (Object Linking and Embedding) for Process Control
OPC UA	OPC Unified Architecture
QMResults	Quality Measurement Results
OSI	Open Systems Interconnection
RAMI4.0	Referenzarchitekturmodell Industrie 4.0
RFID	Radio Frequency Identification
SCADA	Supervisory Control and Data Aquisition
SDK	Software Development Kit
SHDR	Simple Hierarchical Data Representation
STEP NC	Standard for the Exchange of Product Model Data
TCP/IP	Transmission Control Protocol / Internet Protocol
TPM	Total Productive Management
u. a.	unter anderem
URI	Unique Resource Identifier
URL	Unique Resource Locator
URN	Unique Resource Name
UUID	Universally Unique Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language
xmlns	XML Namespace
XSD	XML Schema Definition
z. B.	zum Beispiel