ICS 35.240.50 VDI/VDE/NAMUR-RICHTLINIEN Juni 2017

VEREIN DEUTSCHER INGENIEURE

VERBAND DER ELEKTROTECHNIK ELEKTRONIK INFORMATIONSTECHNIK

INTERESSEN-GEMEINSCHAFT AUTOMATISIERUNGS-TECHNIK DER PROZESSINDUSTRIE

Automatisierungstechnisches Engineering modularer Anlagen in der Prozessindustrie

Allgemeines Konzept und Schnittstellen

VDI/VDE/ NAMUR 2658

Blatt 1

Entwurf

Automation engineering of modular systems in the process industry – General concept and interfaces

Einsprüche bis 2017-09-30

- *vorzugsweise über das VDI-Richtlinien-Einspruchsportal http://www.vdi.de/einspruchsportal*
- *in Papierform an VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik Fachbereich Industrielle Informationstechnik Postfach 10 11 39 40002 Düsseldorf*

Inhalt Seite

VDI-Gesellschaft Mess- und Automatisierungstechnik (GMA)

Fachbereich Industrielle Informationstechnik

VDI-Handbuch Informationstechnik, Band 1: Angewandte Informationstechnik
VDI/VDE-Handbuch Automatisierungstechnik

Zu beziehen durch Beuth Verlag GmbH, 10772 Berlin – Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017

Vervielfältigung – auch für innerbetriebliche Zwecke – nicht gestattet

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017

Vorbemerkung

Der Inhalt dieser Richtlinie ist entstanden unter Beachtung der Vorgaben und Empfehlungen der Richtlinie VDI 1000.

Alle Rechte, insbesondere die des Nachdrucks, der Fotokopie, der elektronischen Verwendung und der Übersetzung, jeweils auszugsweise oder vollständig, sind vorbehalten.

Die Nutzung dieser Richtlinie ist unter Wahrung des Urheberrechts und unter Beachtung der Lizenzbedingungen (www.vdi.de/richtlinien), die in den VDI-Merkblättern geregelt sind, möglich.

Allen, die ehrenamtlich an der Erarbeitung dieser Richtlinie mitgewirkt haben, sei gedankt.

Einleitung

Diese vom Fachausschuss „Zukünftige Architekturen in der Automation“ der VDI/VDE-Gesellschaft für Mess- und Automatisierungstechnik gemeinsam mit der Namur und dem ZVEI erstellte Richtlinie definiert die Spezifikation von Modulschnittstellen zur Verwendung in modularen Anlagen und beschreibt diese syntaktisch, semantisch und pragmatisch.

Modulare Anlagen werden in der Fertigungs- und Verfahrenstechnik vermehrt eingesetzt [6]. Ziel hierbei ist, sowohl die Planungszeit neuer Anlagen als auch die Umbauarbeiten an Anlagen zeitlich deutlich zu verkürzen. Hierdurch reduziert sich die Stillstandzeit bzw. wird die Time-to-Market bei Neuanlagen deutlich verkürzt.

Die Domänen „Fertigungstechnik“ und „Verfahrenstechnik“ stellen hierbei sehr unterschiedliche Anforderungen an die Modularität. In dieser Richtlinie wird vornehmlich die Verfahrenstechnik betrachtet.

Ausgehend von abgeschlossenen Projekten, wie „F3 Factory“ [1], und bestehenden Empfehlungen und Anforderungen an verfahrenstechnische Module – veröffentlicht in der NE 148 – wird in dieser Richtlinie das Engineering der Automatisierungstechnik modularer Anlagen beschrieben. Hierbei wird sowohl das Modulengineering als auch das Anlagenengineering der Automatisierungstechnik betrachtet.

Zur Beschreibung der Modultypen wird das Module Type Package (MTP) verwendet, das die Schnittstellen und Funktionen der Automatisierungstechnik von Modulen definiert und beschreibt und letztlich die Integration von Modulen in eine Prozessführungsebene (PFE) ermöglicht.

In den einzelnen Blättern dieser Richtlinienreihe werden folgende Aspekte fokussiert:

- allgemeines Konzept zum Engineering modularer Anlagen
- Zustands- und Dienstmodelle modularer Anlagen
- konzeptionelle Architektur und Aufbau des MTP
- Strukturierung des MTP anhand verschiedener zu berücksichtigender Aspekte
- Schnittstellen des Verhaltensvertrags zwischen Diensten des Moduls und der PFE
- Definition und Beschreibung des sogenannten *Manifests*, der Verwaltungsdatei des MTP
- Definition und Beschreibung der Kommunikationsschnittstellen eines Moduls am Beispiel OPC UA
- Modellierungsvorschriften zur Erstellung des Manifests und der darin beinhalteten Kommunikationsbeschreibung

Weitere in Vorbereitung befindliche Richtlinien dieser Reihe greifen folgende Aspekte des automatisierungstechnischen Engineerings modularer Anlagen auf:

- Blatt 2: Modellierung von Bedienbildern
- Blatt 3: Bibliothek für Datenobjekte
- Blatt 4: Modellierung von Moduldiensten
- Blatt 5: Laufzeit- und Kommunikationsaspekte

Zusätzlich geplant sind Richtlinien zu den Themen „Diagnose“, „Alarmmanagement“, „funktionale Sicherheit“ sowie „Validieren von MTP und Modulen“.

Eine Liste der aktuell verfügbaren Blätter dieser Richtlinienreihe ist im Internet abrufbar unter www.vdi.de/2658.

Durch die zunehmende Vernetzung der Module werden weitere Themen hinzukommen, z.B. modulübergreifende funktionale Sicherheit oder sichere Kommunikation zwischen Modulen.

1 Anwendungsbereich

Modulare Anlagen werden in der Verfahrenstechnik, insbesondere in der Chemie, Feinchemie, Spezialchemie und pharmazeutischen Industrie verwendet, um schnell eine zur industriellen Produktion verwendbare Anlage zu entwickeln. Hierbei steht der zeitliche Aspekt, das heißt die schnelle Marktreife der Anlage, im Vordergrund.

Ein weiterer Anwendungsfall ist die effiziente Einbindung beispielsweise von Package Units oder Logistiksystemen in Bestandsanlagen oder neue Anlagen.

Daraus ergeben sich nach [1] und [2] im Wesentlichen drei Anwendungsszenarien für modulare Anlagen:

- kurze Time-to-Market

  Individuelle Anlagen schnell aus vorhandenen Modulen aufbauen.
- schnelle Time-to-Repair

  Die Verfügbarkeit von Anlagen erhöhen, indem defekte Module schnell gegen Gleichwertige ausgetauscht werden.
- individuelle Produktion kleiner Chargen

  Anlagen durch Austausch von Modulen effizient umbauen.

Hierfür werden verfahrenstechnische Module im Sinne der NE 148 verwendet. Diese einfachen, in sich abgeschlossenen Module lassen sich mit geringem Aufwand zu komplexen Gesamtanlagen verschalten.

In dieser Richtlinie werden Anwendungsfälle zur Automatisierung von Modulen und modularen Anlagen betrachtet und die Anforderungen der Module, beschrieben in der NE 148, verwendet. Deshalb setzt sich die Zielgruppe dieser Richtlinie aus verschiedenen Interessengruppen zusammen:

- Modulhersteller

  Der Modulhersteller kann anhand dieser Richtlinie die Automatisierung seines Prozessmoduls kompatibel zu beliebigen Systemen der Prozessführungsebene beschreiben, damit diese im Sinne der NE 148 in eine modulare Anlage integriert werden kann.
- Werkzeughersteller

  Bei Werkzeugherstellern wird grundsätzlich in zwei Kategorien unterteilt:

  1) Werkzeughersteller, der ein Projektierungswerkzeug für Module entwickelt

     Diesem wird in der Richtlinie Hilfestellung bezüglich des nötigen Informationsgehalts und der MTP-Exportschnittstelle für sein Werkzeug gegeben.
  2) Werkzeughersteller für PFE-Integrationswerkzeuge

     Diesem werden die Inhalte des MTP nahegelegt und Interpretationshilfen semantischer und syntaktischer Natur zur Seite gestellt.
- Modulintegrator

  Zur Integration werden MTPs verwendet und letztlich wird der Modulintegrator – aufgrund der Funktionalität der Module – entscheiden, welche Module eingesetzt werden sollen. Die funktionale Beschreibung findet ebenfalls im MTP statt, weshalb auch der Modulintegrator Nutzen aus dieser Richtlinie zieht.

Es ist nicht ausgeschlossen, dass beispielsweise der Modulintegrator auch gleichzeitig Modulhersteller sein kann. Auch andere Mischformen der Zielgruppen sind möglich. Der Anlagenbetreiber wiederum kann selbstverständlich auch eine der oben genannten Rollen einnehmen.

Aus Abschnitt 3 wird ersichtlich, dass nicht alle Modultypen, Modulvarianten und Transparenzstufen fokussiert werden können. Der Schwerpunkt dieser Richtlinie liegt deshalb auf integrierbaren Modulen mit eigener Intelligenz und Greybox-Transparenzstufe, die aus verfahrenstechnischer Sicht als modulare Funktionseinheit bezeichnet wird [6].

Einige der weiteren Varianten, Typen und Transparenzstufen werden darüber hinaus mitbetrachtet. Es ist jedoch nicht gegeben, dass die beschriebenen Konzepte und Austauschformate vollumfänglich mit den anderen Varianten, Typen und Transparenzstufen kompatibel sind.

Des Weiteren wird davon ausgegangen, dass integrierbare Module in sich geschlossen und gekapselt sind und die Abhängigkeiten zwischen Modulen damit auf ein Minimum reduziert sind. Dies erfordert, dass die Module sich selbst und ihre Umgebung schützen.

Zukünftig ist es denkbar, dass Module zunehmend untereinander kommunizieren, um kollaborative Funktionen (auch sicherheitsgerichtet) implementieren zu können oder Mehrfachinstrumentierung zu vermeiden.

2 Abkürzungen

In dieser Richtlinie werden die nachfolgend aufgeführten Abkürzungen verwendet:

HMI	Mensch-Maschine-Schnittstelle (Human Machine Interface)
IC	Interface Class
MTP	Module Type Package
PEA	modulare Funktionseinheit (Process Equipment Assembly)
PFE	Prozessführungsebene
PLT	Prozessleittechnik
SUC	System Unit Class

3 MTP Versionierung

DocumentIdentifier: VDI/VDE/NAMUR 2658-1

Major.Minor.Patch: 1.0.0

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017

4 Module

Es gilt, drei Definitionen bezüglich der in dieser Richtlinie betrachteten Module zu treffen:

a) die Modulvarianten nach NE 148
b) die Modultypen
c) die betrachtete Transparenzstufe

4.1 Modulvarianten

Die NE 148 definiert drei Modulvarianten:

- autonome Module

  Diese zeichnen sich dadurch aus, dass sie geschlossene Einheiten bilden. Sie werden an Ver- und Entsorgungseinrichtungen angeschlossen, auch gegebenenfalls in eine PFE integriert, sind jedoch durch diese nicht direkt steuerbar und erfüllen ihre Funktion vollkommen autark. Ein Engineering auf PFE-Seite ist deshalb nicht nötig. Daher muss eine vollständige Automatisierung dieser Modulvariante erfolgen.

- integrierbare Module / modulare Funktionseinheit

  Diese werden funktional vollständig spezifiziert und sowohl energetisch und stofflich als auch automatisierungstechnisch integriert. Die vorgegebenen Funktionen werden durch eine übergeordnete PFE orchestriert und damit in einen Gesamtverbund aus Modulen eingebunden. Diese sollen weitgehend automatisch in eine vorhandene oder neue PFE integrierbar sein.

- modulare Module

  Diese werden innerhalb eines Moduls ebenfalls durch Module aufgebaut. Das heißt: In die erste Modulebene wird eine zweite Modulebene integriert. Das Gesamtmodul ist wiederum in eine PFE integrierbar, wie die integrierbaren Module. Hierdurch sind Mischformen aus integrierbaren Modulen und modularen Modulen realisierbar.

In dieser Richtlinie wird vornehmlich auf die Modulvariante „integrierbare Module“ fokussiert, da davon auszugehen ist, dass diese Variante die weiteste Verbreitung finden wird. Jedoch sind die hierin beschriebenen Konzepte weitgehend änderungsfrei auch auf modulare Module anwendbar.

4.2 Modultypen

Des Weiteren unterscheidet die NE 148 für die Modulvariante „integrierbare Module“ zwischen zwei verschiedenen Modultypen:

- Module mit eigener Intelligenz

  Innerhalb des Moduls befindet sich eine automatisierungstechnische Steuerung, die die Funktion des Moduls weitgehend in Eigenregie abbildet. Die Steuerung stellt die im Modul realisierten verfahrenstechnischen Funktionen über eine Schnittstelle bereit, sodass diese in einer PFE orchestriert (gestartet, gestoppt, pausiert usw.) werden können. Hierdurch wird das Verfahren der Anlage durch eine Abfolge der verschiedenen Funktionen realisiert.

- Module ohne eigene Intelligenz

  Diese Module erhalten eine Remote I/O, um Zugriff auf die innerhalb des Moduls liegenden verfahrenstechnischen Einrichtungen zu erhalten. Die Automatisierungstechnik liegt, wie in konventionellen Anlagen, innerhalb der PFE und wird nicht vom Modul selbst realisiert. Es ist denkbar, dass die verfahrenstechnisch möglichen Funktionen des Moduls formal beschrieben dem Modul beiliegen, sodass diese in der PFE automatisch erzeugt werden können, auf eine generische Schnittstelle direkt zum Modul wird jedoch weitgehend verzichtet.

Diese Richtlinie fokussiert auf Module mit eigener Intelligenz. Der Modultyp „Module ohne eigene Intelligenz“ kann durch konventionelle Methoden beschrieben werden.

4.3 Transparenzstufen von Modulen

Neben den zuvor beschriebenen Modultypen und Modulvarianten lassen sich Module durch ihre Transparenzstufe charakterisieren. Hierbei wird von Black-, Grey- und White-Box-Integration gesprochen. Im Folgenden werden die Transparenzstufen definiert. Die Definition ist angelehnt an [3]:

- Blackbox-Module

  Diese stellen die von ihnen angebotenen Dienste nach außen zur Verfügung. Diese können in ein übergeordnetes System integriert und von diesem angesprochen werden. Die innerhalb des Moduls verwendeten PLT-Stellen werden nicht offengelegt. Ebenso wird kein Bedienbild auf Basis der PLT-Stellen bereitgestellt, sondern lediglich eine Visualisierung (gegebenenfalls nur über Zustandsvariablen) der Zustände der Dienste. Die Logik des Moduls wird dem Anwender ebenfalls verborgen. Diagnosemöglichkeiten sind in diesem Darstellungstyp dem Modulhersteller vorbehalten.

- Greybox-Module

  Diese stellen genau wie Blackbox-Module die angebotenen Dienste nach außen zur Integration in eine PFE zur Verfügung. Zusätzlich zu den Diensten werden Bedienbilder bereitgestellt, die benötigt werden, um das Modul in der Anlage zu fahren. Darum werden in Grey-Box Modulen auch PLT-Stellen offengelegt. Welche PLT-

Stellen im Einzelnen offengelegt werden und somit in der PFE sichtbar sind, entscheidet der Modulhersteller: Dieser publiziert über das MTP nur die PLT Stellen, die sichtbar sein sollen. Dies gilt auch für eventuell vorhandene Betriebsartenumschaltungs- und Verriegelungsinformationen. Eine Diagnose kann auf Modulebene stattfinden. Die PLT-Stellen stellen keine eigenen Diagnosefunktionen bereit. Die Logik des internen Automatisierungssystems wird bei Greybox-Modulen nicht offengelegt.

- Whitebox-Module

  In Whitebox-Module kann beliebig tief hineingeschaut werden, da alle PLT-Stellen vollständig beschrieben sind. Der Zugriff wird auf alle Teile des Moduls lesend und schreibend gewährt, dies beinhaltet auch die Logik der integrierten Steuerung. Dennoch können auch Whitebox-Module die angebotenen Dienste nach außen zur Integration in eine PFE zur Verfügung stellen.

Im Fokus des in dieser Richtlinie beschriebenen Module Type Package sowie der Laufzeitintegration von Modulen liegt die Greybox-Integration, da davon auszugehen ist, dass diese am häufigsten Anwendung findet.

5 Grundkonzepte der Automatisierung modularer Anlagen

Modularisierung stellt einen erheblichen Schritt zur Beschleunigung von Bau und Planung von Produktionsanlagen kleiner bis mittlerer Größe dar. Durch die Verwendung von branchenspezifischen Modulen, die fertige, in sich geschlossene funktionale Einheiten bilden, kann der Engineeringaufwand für Prozessanlagen erheblich reduziert werden. Die Wiederverwendung von bewährten Modullösungen vermeidet Fehler in frühen Phasen des Engineerings, was wiederum zu Kosten- und Zeitersparnissen führt. Neben der angestrebten Skalierbarkeit durch Hinzu- und Entnahme von Modulen (Numbering up) wird auch eine bessere Datenbasis zur Kostenschätzung für die Planung und Errichtung der Anlagen geschaffen, da übliche Aufwände sich lediglich in neuer Zusammensetzung wiederholen.

5.1 Engineering Workflow

Betrachtet man das Engineering einer modularen Anlage, so muss grundsätzlich zwischen zwei Engineering-Phasen unterschieden werden [5] (siehe auch Bild 1):

- Modulengineering

  Jede modulare Funktionseinheit (PEA) wird einmalig entwickelt (siehe Bild 1, linke Seite). Beinhaltet ist sowohl die physikalische Gestaltung des hierin zu realisierenden Verfahrensschritts, als auch die informationstechnische Schnittstelle zu übergeordneten Systemen. Hinzu kommt das Engineering der Steuerungslogik und der Bedienbilder. Das Engineering entspricht dem einer kleinen Prozessanlage, mit dem Unterschied, dass das Modul in seiner Funktion generisch für verschiedene Einsatzarten ausgelegt werden sollte. Die im Engineering erzeugten Daten über Aufbau, die vom Modul bereitgestellten Informationsschnittstellen, Prozessabläufe, sogenannte Dienste und die dazugehörigen Bedienbilder werden im MTP abgelegt.

- PFE-Engineering

  Beim PFE-Engineering werden die modularen Funktionseinheiten (im Weiteren verkürzt Modul genannt) in die PFE integriert (siehe Bild 1, rechte Seite). Letztlich wird die Modulbeschreibung in die PFE eingelesen und die nötige Konfiguration auf PFE-Seite für jedes benötigte Modul generiert. Dies beinhaltet sowohl die Schnittstelle zum Informationsmodell des Moduls als auch die grafische Darstellung desselben in der PFE. Zusätzlich wird der modulübergreifende verfahrenstechnische Prozess in der PFE konfiguriert. Die Prozedursteuerung zum zeitgerechten Abrufen und Überwachen (Orchestrierung) der Moduldienste wird erstellt. Die Ansteuerung der Dienste der Module wird parametriert und gegebenenfalls modulübergreifende Verriegelungslogik erzeugt. Letztlich wird die physikalische Kommunikation im Netzwerkengineering abgebildet und parametriert.

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017

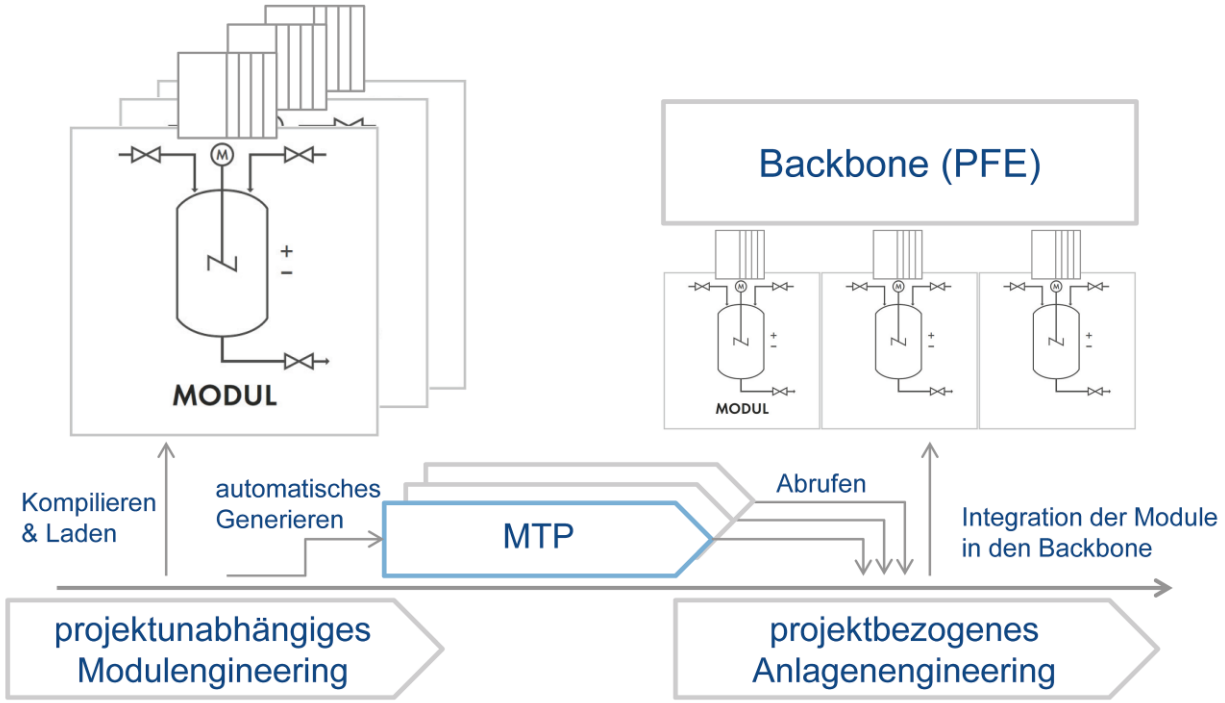


Bild 1. Engineeringphasen modularer Anlagen (Quelle: NAMUR/ZVEI)

Dabei sind die Engineeringprozesse des Modulherstellers (Modulengineering) und Modulintegrators (PFE-Engineering) voneinander entkoppelt. Der Mehrwert der Modulhersteller liegt in der Mehrfachanwendung der funktionalen Prozesseinheiten; der Mehrwert des Modulintegrators liegt in der verringerten Engineeringzeit für die Gesamtanlage.

5.2 Funktionale Modularität

In Analogie zur physikalischen Modularisierung der Anlage nach NE 148 erfolgt eine funktionsorientierte Modularisierung. Dabei stellt ein Modul seine verfahrenstechnischen Funktionen als Dienste einer übergeordneten PFE zur Verfügung. Es nimmt damit die Stellung eines Diensteanbieters ein. Die vom Modul angebotene Dienstleistung kann von der PFE abgerufen werden, die damit ein Dienstnutzer ist.

5.3 Dienstbasierte Steuerung

Um die Dienste der einzelnen Module in eine für die Produktion des gewünschten Produkts erforderliche Folge zu bringen (Orchestrierung), muss z.B. bei einem kontinuierlich betriebenen Reaktionsprozess das Anfahren des Reaktors mit dem Vorlegen der Ausgangsprodukte abgestimmt werden. Da diese zusätzliche Orchestrierungsfunktion erst durch Kombination verschiedener Module notwendig wird, muss diese Funktion von einer noch während des PFE-Engineering gestaltbaren Automatisierungsinstanz, z.B. dem übergeordneten Leitsystem, übernommen werden.

Die in den Modulen vorgesehenen prozesstechnischen Funktionen werden als Dienste gekapselt. So könnte beispielsweise ein Reaktormodul mit einem Mixer den Dienst „Mischen“ anbieten. Da die Edukte in den Reaktor einzufüllen sind, wird vom Reaktor des Weiteren der Dienst „Befüllen“ angeboten, der sich je nach Anzahl und Benennung der Einfüllstutzen z.B. in „BefüllenA“ und „BefüllenB“ unterscheiden kann. Verfügt der Reaktor über ein Heizsystem, kann ebenfalls der Dienst „Heizen“ ausgeführt werden. Ein entsprechender Parametersatz dieses Diensts könnten die Zieltemperatur, die Temperatur-Anstiegsgeschwindigkeit und die Haltezeit sein.

Hierbei wird jeder Dienst durch ein Zustandsmodell abgebildet, das in VDI/VDE 2658 Blatt 4 näher beschrieben wird.

5.4 Bedienerschnittstelle für modulare Anlagen

Die zentrale Herausforderung der Bedien- und Beobachtbarkeit des über mehrere Module verteilten Prozesses ist sowohl die automatische Bedienbilderstellung als auch die Realisierung des nach NE 148 formulierten einheitlichen „Look-and-feel“ einer modularen Anlage.

Da der Modulhersteller für die Planung, den Aufbau und die Programmierung des Moduls verantwortlich ist, entwirft er mit den in Blatt 2 und Blatt 3 dieser Richtlinienreihe definierten Objekten ein oder mehrere Bedienbilder. Die Modellierung der Bedienbilder wird in Blatt 2 dieser Richtlinienreihe

fokussiert. Eine Kenntnis der in industriellen Anlagenprojekten meist projektspezifisch verwendeten Bedienbildbibliothek der übergeordneten PFE hat er zu diesem Zeitpunkt jedoch nicht. Die Generierung des modulspezifischen Bedienbilds in der PFE der Gesamtanlage kann somit erst nach Integration des Moduls im PFE-Engineering erfolgen. Während des Imports in die PFE werden die generischen HMI-Objekte durch konkrete, zielsystemspezifische Symbole ersetzt.

Zur Umsetzung der modulspezifischen Bedienbilder in projektspezifisch vereinheitlichten Bedienbildelementen müssen die Bedienbilder in einer darstellungsunabhängigen Beschreibungsform vorliegen. Diese Beschreibung enthält die Information zum Typ des darzustellenden Prozessequipments sowie dessen Lage- und Größeninformation. Diese Informationen sind durch einen Algorithmus auswertbar, der die projektabhängigen Bedienbildelemente in gewünschter Darstellung und Lage auf das Bedienbild in der PFE setzt und mit den entsprechenden Variablen für die Kommunikation mit der Modulsteuerung verknüpft.
Die Nutzung eines Bibliothekskonzepts bedingt, dass die Bibliothek sowohl im Engineeringwerkzeug des Modulherstellers als auch im Engineeringwerkzeug für die PFE vorliegt. Die Bibliothek ist Bestandteil von Blatt 3 dieser Richtlinie.

Alle Informationen, die im Modulengineering erarbeitet und während des PFE-Engineerings benötigt werden, werden in der in Abschnitt 5 beschriebenen Modulbeschreibung abgelegt. Mithilfe dieses Informationsträgers wird die PFE entsprechend aufgesetzt.

5.5 PFE-Integration

Werden mehrere Module zu einer Anlage kombiniert und physikalisch verbunden, entsteht die Notwendigkeit, diese Module im Rahmen des PFE-Engineerings in eine übergeordnete PFE zu integrieren (Bild 2). Die PFE muss gewährleisten, dass alle zu integrierenden Objekte eines Moduls innerhalb des Namensraums der PFE eindeutig unterscheidbar sind, beispielsweise durch eine modulspezifische Präfixvergabe.

Als PFE kann beispielsweise ein Prozessleitsystem eingesetzt werden. Als Kommunikationstechnologie lässt sich z.B. OPC UA einsetzen, wobei sich der OPC UA Server des Moduls direkt auf der Steuerung des Moduls beziehungsweise im Modul befindet und durch den Modulhersteller erstellt und parametriert wird. In diesem sind alle – für die Kommunikation zwischen Modul und PFE notwendigen – Informationen abgebildet.

OPC UA dient hier als Beispiel, ist im Rahmen dieser Richtlinie aber nicht gesetzt und kann durch andere Kommunikationsmechanismen ersetzt oder erweitert werden. Dies thematisiert Blatt 5 dieser Richtlinie.

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017

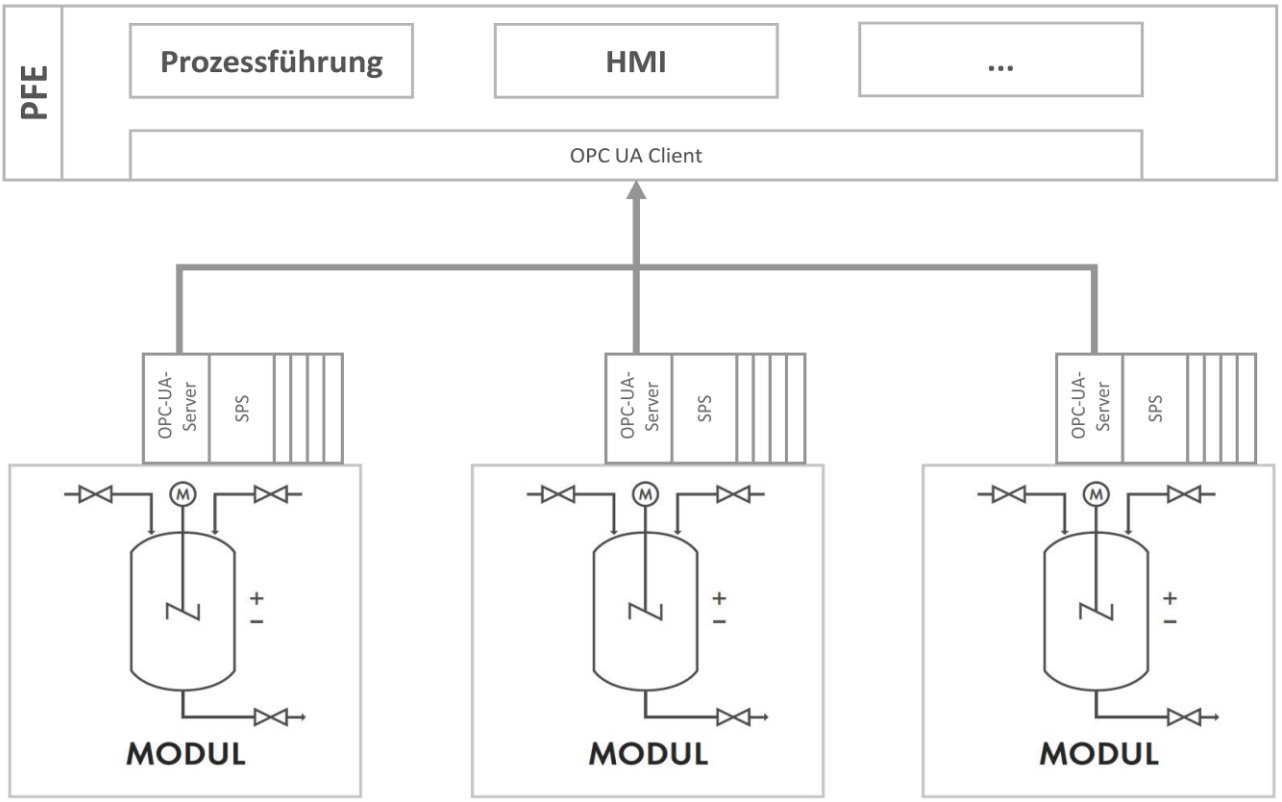


Bild 2. Beispielarchitektur zur Integration von Modulen in eine Prozessführungsebene (PFE) über OPC UA

5.6 Security modularer Anlagen

Bezüglich der Informationssicherheit in industriellen Anlagen wird auf die Richtlinienreihe VDI/VDE 2182 verwiesen, die ein allgemeines Vorgehensmodell für die Informationssicherheit in der industriellen Automatisierung beschreibt.

5.7 Funktionale Sicherheit modularer Anlagen

Bezüglich der funktionalen Sicherheit in industriellen Anlagen wird auf DIN EN 61508 sowie DIN EN 61511 sowie auf VDI/VDE 2180 verwiesen.

In dieser VDI-Richtlinienreihe wird auf die im MTP abzubildenden Aspekte der funktionalen Sicherheit eingegangen.

6 Module Type Package

Das Module Type Package (MTP) ist das zentrale Modulbeschreibungs- und Datenaustauschformat. Das MTP beschreibt die zur Integration in die PFE notwendigen Modulschnittstellen und -funktionen. Die Modellierung des MTP findet angelehnt an [4] auf Basis von AutomationML (Automation Markup Language, siehe IEC 62714) statt. Hierbei gibt es einige Einschränkungen, so wird zum Beispiel auf die Verwendung von Rollenklassen verzichtet.

6.1 Verwendung bestehender Standards

Zur Integration während des Engineerings werden verschiedene Aspekte betrachtet. Ausgehend vom dienstorientierten Konzept der Modulsteuerung, werden die angrenzenden Bereiche „Bedienbilder“, „Diagnose“, „Historie“, „Zustandsmodelle“ usw. betrachtet (siehe Bild 3).

Angefangen bei der Feldebene besitzt jedes Modul PLT-Stellen, um Mess-, Steuer- und Regelaufgaben zu erfüllen. Diese sind allerdings im Greybox-Modul, wie es als Grundlage für das hier beschriebene Modellierungsverfahren verwendet wurde, zumindest teilweise und in entsprechend geminderten Umfang in der PFE sichtbar.

Bedienbilder werden mithilfe von AutomationML abgebildet. Dabei wird das Quellbedienbild im Engineeringwerkzeug des Modulherstellers so beschrieben, dass es im HMI-Werkzeug der PFE automatisch generiert werden kann. Dazu müssen über alle Bestandteile des Quellbedienbilds Informationen zur Lage, Größe, Vorgänger-/Nachfolgerbeziehungen und Bedeutung übertragen werden.

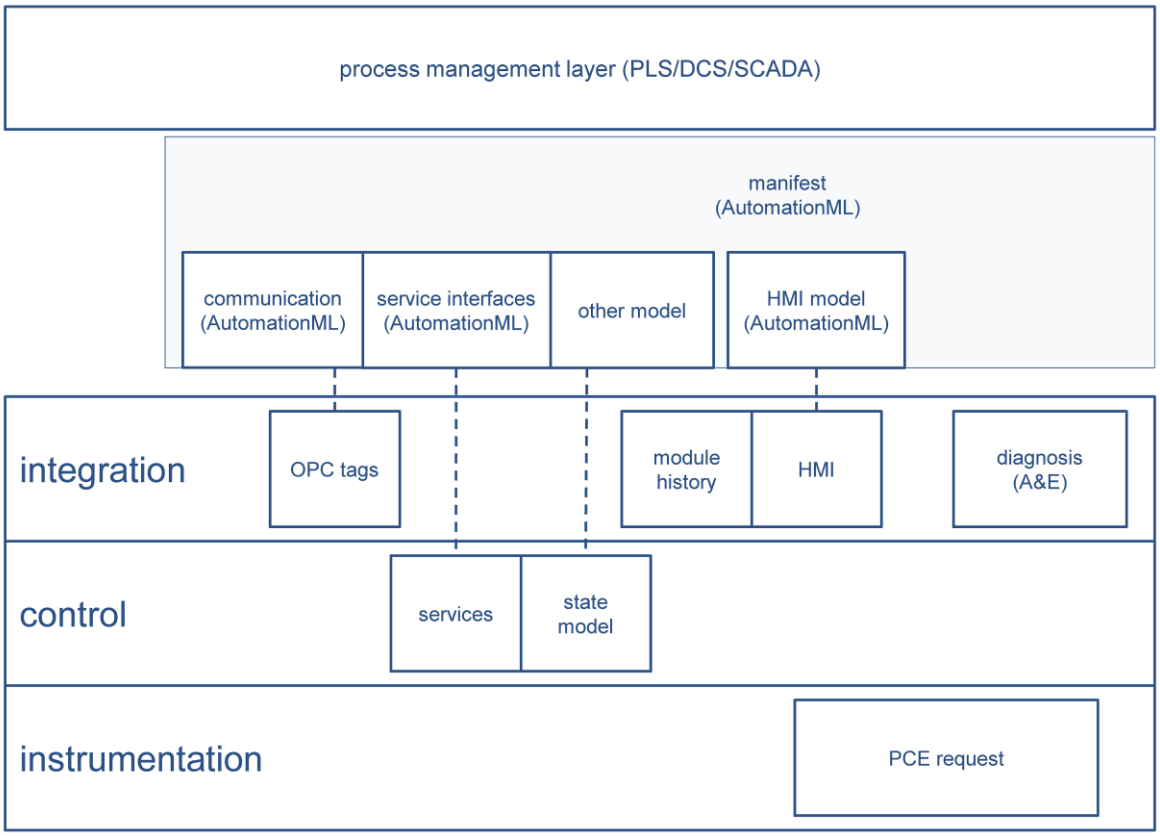


Bild 3. Aspekte eines MTP und deren Zusammenwirken für verschiedene Aufgaben in der PFE

Die hier abgebildeten Aspekte dienen als Beispiele und sind nicht als vollständig zu betrachten.

Das Modul ist für die Sicherung seiner eigenen Modulhistorie verantwortlich, ggf. auch unter Nutzung eines externen Historiensystems. Im Gegensatz zu konventionellen Anlagen muss die Historie der Module der PFE nicht bekannt sein. Einzelne Aspekte der Modulhistorie sollten jedoch der PFE bei der Integration des Moduls zur Verfügung gestellt werden können.

Die eigentliche Steuerung, das heißt, die Schnittstelle zu den bereitgestellten Diensten, wird über eine AutomationML-Beschreibung zur Verfügung gestellt. Diese Informationen müssen der PFE bekannt sein, um später zur Laufzeit die Dienste zu orchestrieren, das heißt in eine dem Produktionsprozess entsprechende geordnete Reihenfolge zu bringen.

Um Überschneidungen bei Dienstanfragen zu vermeiden, erhält jeder Dienst ein ihm zugeordnetes Zustandsmodell, das ebenfalls im MTP beschrieben wird.

Durch die im Folgenden beschriebene Struktur ist es außerdem möglich, Beschreibungen von anderen Aspekten, die nicht innerhalb der Spezifikation des MTP liegen, einzubinden. So besteht beispielsweise die Möglichkeit, IEC-61131-3-Code als PLCOpen XML abzubilden. Die Integration erfolgt dann über die von AutomationML bereitgestellten Schnittstellen.

6.2 Aufbau des Module Type Package

Das MTP weist eine offene Architektur auf. Das Grundkonzept sieht eine Organisationsdatei, das sogenannte *Manifest*, vor, das als Inhaltsverzeichnis des MTP anzusehen ist. Dies verweist auf die verschiedenen Aspekte im MTP, sodass eine einfache Navigation durch das komplette MTP gegeben ist, siehe Bild 4.

Wird ein weiterer Aspekt in das MTP aufgenommen, so wird auch hier wieder ein Verweis auf diesen erzeugt. Die anderen Aspekte sind davon nicht betroffen, und so können Aspekte nach und nach in das MTP aufgenommen werden.

Jeder Aspekt kann in einem dafür vorgesehenen Format abgelegt werden. Wird für einen Aspekt ebenfalls AutomationML verwendet, so kann dieser auch in der gleichen Datei wie das Manifest enthalten sein. Der im Manifest enthaltene Verweis wäre in diesem Fall auf eine weitere *InstanceHierarchy* innerhalb der gleichen AutomationML-Datei – anstelle eines relativen Verweises auf eine andere Datei – zu realisieren.

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017

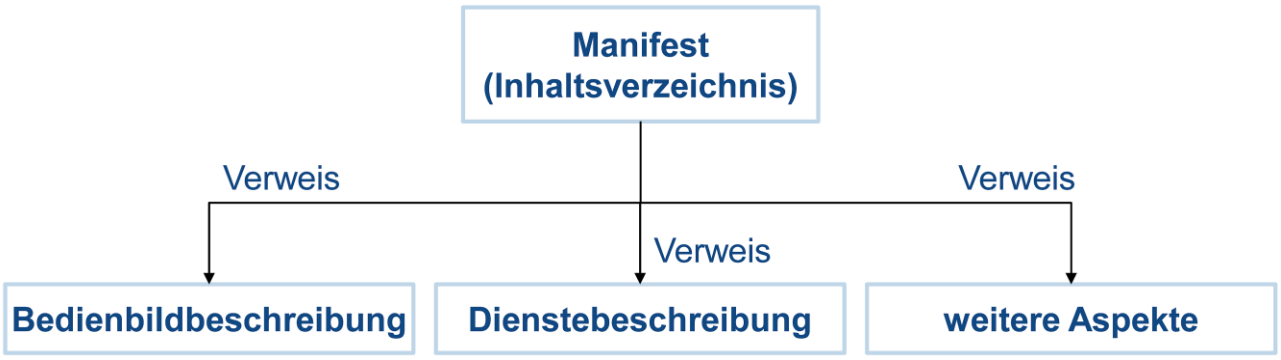


Bild 4. Manifest als Inhaltsverzeichnis

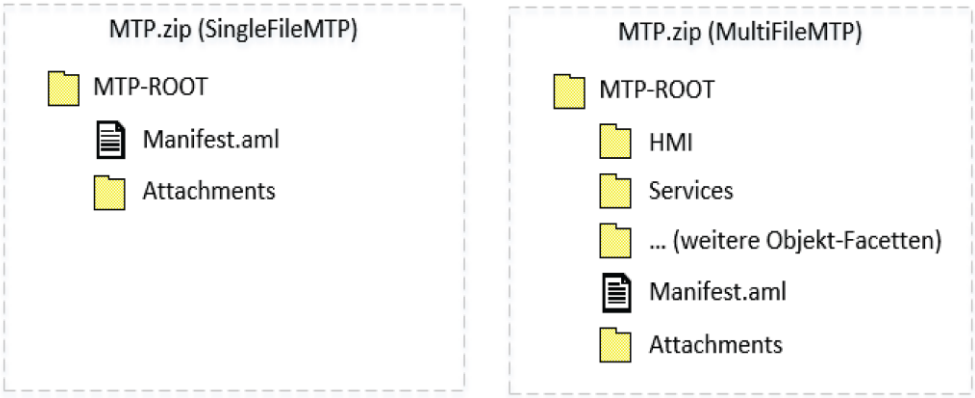


Bild 5. Packaging von MTPs

Zusätzlich zum Inhaltsverzeichnis wird im Manifest die Kommunikation zum Modul beschrieben. Die Kommunikation kann aus verschiedenen Aspekten heraus adressiert werden und stellt die informationstechnische Verbindung zur Modulsteuerung dar.

6.3 MTP-Packaging-Format

Ein Module Type Package wird grundsätzlich als ZIP-komprimierte Datei mit der Dateiendung *.mtp oder *.zip transportiert. Der MIME-Type eines MTP ist "application/mtp".

Die innere Struktur eines Module Type Package kann sich allerdings unterscheiden. Die Informationen, die in Form von AutomationML vorliegen, können in Form einer einzelnen Datei mit mehreren *InstanceHierarchies* oder als getrennte Dateien, in denen jeweils eine *InstanceHierarchy* modelliert ist, organisiert sein.

Die Variante mit einer Datei und mehreren *InstanceHierarchies* wird im Folgenden *SingleFileMTP* genannt. Die zweite Möglichkeit heißt *MultiFileMTP*.

- *SingleFileMTP*

  Für einen SingleFileMTP wird die in Bild 5 links dargestellte Ordnerstruktur definiert. Da alle Dateien im *SingleFileMTP* in der gleichen AutomationML Datei transportiert werden, ist der Umfang der Dateien im Root-Verzeichnis des MTPs gering.

- *MultiFileMTP*

  Für einen *MultiFileMTP* wird die in Bild 5 rechts dargestellte Ordnerstruktur definiert. Die hier, im Vergleich zum *SingleFileMTP*, vorhandenen zusätzlichen Ordner beinhalten die weiteren AutomationML-Dateien, die wiederum die *Instance Hierarchies* der separierten Beschreibungsbestandteile beinhalten. Für jede separierte *InstanceHierarchy* wird ein Unterordner angelegt, in dem die entsprechende AutomationML-Datei abgelegt wird.

Das MTP-Packaging-Format besitzt die Möglichkeit zum Transport weiterer Dateien mit beliebigem Inhalt. Diese werden im Ordner *Attachments* zusammengefasst. Ein Beispiel für einen möglichen Inhalt dieses Ordners sind Handbücher oder Zeichnungen des Moduls.

6.4 Manifest

Das MTP besteht aus mindestens einer *InstanceHierarchy*, dem Inhaltsverzeichnis der Aspekte. Für jeden Aspekt gibt es maximal eine *InstanceHierarchy*. Die *InstanceHierarchies* können entweder

in mehreren AutomationML Dateien aufgeteilt werden, oder in einer AutomationML Datei zusammengefasst werden. Bei beiden Möglichkeiten werden *ExternalInterfaces* verwendet, die entweder dateiintern auf die zugehörige *InstanceHierarchy* oder mittels einer URL auf eine externe Datei verweisen.

Das Manifest ist die *InstanceHierarchy* mit dem Namen „ModuleTypePackage“. Die *InstanceHierarchies* der anderen Aspekte können beliebig benannt werden. Innerhalb der *InstanceHierarchy* des Manifests wird genau ein *InternalElement* vom Typ *ModuleTypePackage* instanziiert, das als Namen den Modulnamen erhält. Unterhalb dieses *InternalElements* wird das Inhaltsverzeichnis aufgebaut.

In diesem *InternalElement* vom Typ *ModuleTypePackage* wird die Version der MTP-Instance durch ein Attribut mit dem Namen *Version* abgebildet.

Neben der Version der MTP-Instanz soll auch die entsprechende Version der Richtlinie, nach dessen Inhalt der MTP aufgebaut ist, angegeben werden. Dies ist sowohl für das Manifest als auch für die in AutomationML modellierten Aspekte zu geschehen. Die Versionierung erfolgt nach dem Schema Major.Minor.Patch. In jedem Richtlinienblatt werden die konkreten Werte in Abschnitt 3 definiert.

Die im Manifest verwendeten *SystemUnitClasses*, die zur Erstellung des Inhaltverzeichnisses und der Kommunikation verwendet werden, sind in Bild 6 dargestellt. Wie dort gezeigt, kann das Manifest später um weitere Aspekte erweitert werden.

Die Klassen der Aspekte werden in den weiteren Blättern dieser Normenreihe spezifiziert. Hinzu kommt die Klasse *CommunicationSet*, dessen Instanz integraler Bestandteil des Manifests ist, da viele Aspekte diese benötigen.

6.4.1 Erstellen des Inhaltsverzeichnisses

Innerhalb des Manifests wird das Inhaltsverzeichnis für die verschiedenen im MTP abgebildeten Aspekte abgelegt. Hierfür steht eine Klasse *MTPSet* zur Verfügung, die als Mutterklasse dient und von welcher Klassen abgeleitet werden können. Diese wiederum werden, hierarchisch unter dem *InternalElement* des Typs *ModuleTypePackage* durch *InternalElements* instanziiert. Jedes *InternalElement*, dessen Typ von der Klasse *MTPSet* abgeleitet ist, stellt einen Eintrag im Inhaltsverzeichnis des Manifests dar. Hierdurch kann auf die verschiedenen Aspekte des MTP verwiesen werden.

Innerhalb der Verzeichniseinträge befindet sich immer ein *ExternalDataConnector* aus der *AutomationMLInterfaceClassLib*-Bibliothek, dessen Attribut *refURI* als Wert (Value) der relative Pfad zur Beschreibungsdatei des spezifischen Aspekts oder der Name der zugehörigen *InstanceHierarchy* innerhalb der Datei des Manifests ist. Hierbei unterscheidet sich der Wert des *AttributeDataType* des Attributes *refURI* des *ExternalDataConnectors*. Verweist der relative Pfad zu einer Beschreibungsdatei des Aspektes, so ist der Wert *xs:anyURI*. Wird auf eine *InstanceHierarchy* innerhalb der Datei des Manifests verwiesen, ist der Wert *xs:IDREF*.

Ein Verzeichnis mit drei Einträgen wird beispielhaft in Bild 7 dargestellt, wobei der Aspekt *Communication* für jeden Modultyp zwingend enthalten sein muss. Der Eintrag *Services* zeigt hierbei auf eine externe AutomationML-Datei, die sich relativ zum Manifest im Unterordner *Services* befindet. Der Eintrag *HMIs* zeigt auf eine weitere *InstanceHierarchy* innerhalb der AutomationML-Datei, die das Manifest beinhaltet, mit dem Namen „HMI“.

Eine Ausnahme dieser Architektur bildet das *InternalElement* vom Typ *CommunicationSet*. Da die Kommunikation zum Modul ebenfalls im Manifest abgebildet wird, ist hierfür eine andere Struktur notwendig.

6.4.2 Modellierung der Kommunikation

Die Kommunikation zum Modul wird ebenfalls im Manifest beschrieben. Hierfür steht die SUC *CommunicationSet* zur Verfügung. Die Klasse *CommunicationSet* wird durch zwei *InternalElements* strukturiert: Ein Element vom Typ *SourceList* beschreibt die Datenquellen des Moduls, z.B. einen OPC UA Server, ein weiteres vom Typ *InstanceList* fasst die Instanzen der verwendeten Objekte zusammen, z.B. ein im Modul verwendetes Ventil.

Jede Datenquelle des Modultyps wird von der SUC *ServerAssembly* abgeleitet. Eine Instanz eines *ServerAssembly* wird in die *SourceList* des *CommunicationSet InternalElement* eingefügt. Unterhalb des *ServerAssembly*-Objekts werden die durch die spezifische Datenquelle bereitgestellten Variablen spezifiziert.

Bild 6 stellt beispielhaft die Modellierung von Datenquellen dar. Als Datenquelle kommt hier ein OPC UA Server zum Einsatz. Die Klasse *OPCUAServer* besitzt ein AutomationML-Attribut *Endpoint*, das als Wert den OPC UA Endpoint des zugehörigen Servers bekommt. Hierdurch kann später automatisch eine Kommunikation zum Modul hergestellt werden.

Unterhalb der Datenquelle, im Beispiel dem *OPCUAServer*, wird für jeden Zugriffspunkt ein *ExternalInterface* vom Typ *DataItem* oder einer hiervon spezialisierten *InterfaceClass*, z.B. *OPCUAItem* angelegt, siehe Bild 6.

Ein mit VDI/VDE/NAMUR 2658 Blatt 1 kompatibler MTP muss daher mindestens die Libraries

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017

MTPDataObjectSUCLib mit der SUC *DataAssembly*, *MTPCommunicationSUCLib* mit der SUC *ServerAssembly* und *MTPCommunicationICLib* mit dem Interface *DataItem* gemäß Bild 6 umsetzen.

In Anhang A ist ein entsprechendes Beispiel aufgeführt.

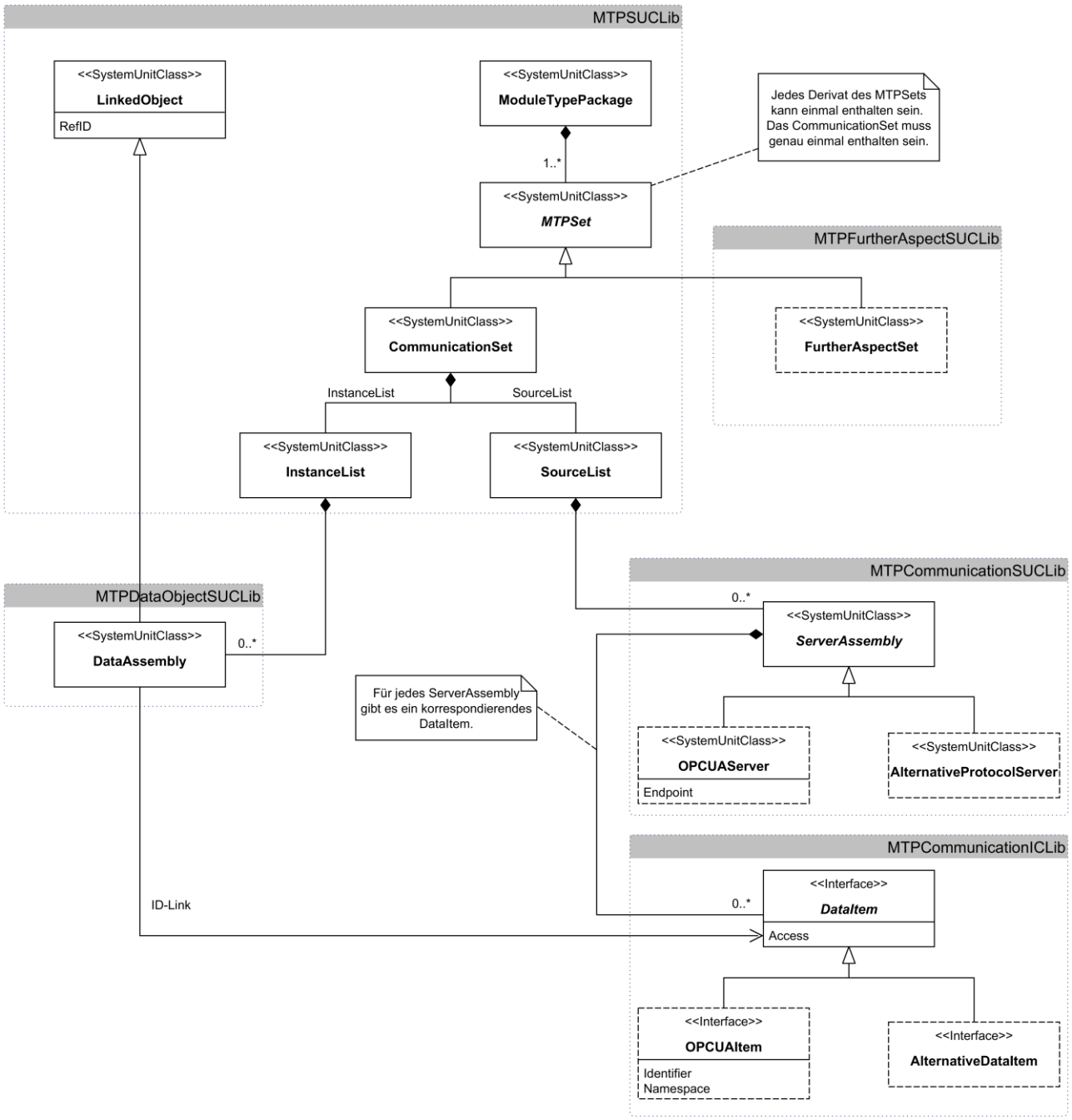


Bild 6. Basis-*SystemUnitClasses* des Module Type Package

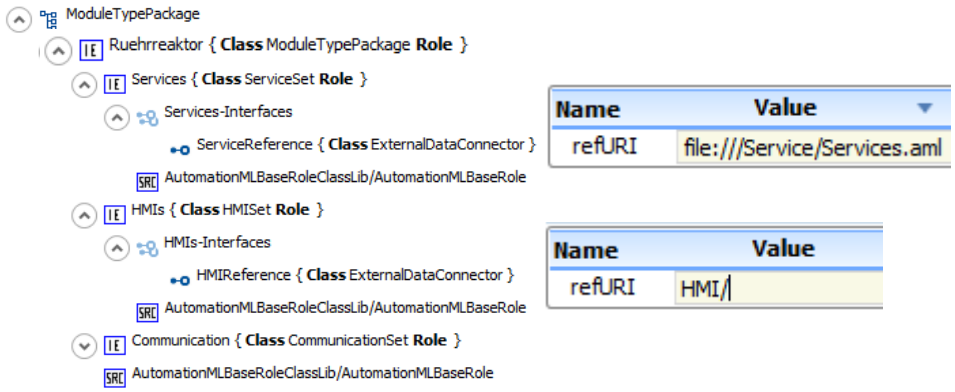


Bild 7. Beispiel eines Inhaltsverzeichnisses im Manifest

Eine *DataItem*-Instanz erhält immer das AutomationML-Attribut *Access*. *Access* beschreibt hierbei die Zugriffsrechte des Zugriffspunkts. Hierbei sind die Attributwerte wie in Tabelle 1 möglich.

Tabelle 1. Mögliche Werte des Attributs *Access*

Attributwert von *Access*	**Bedeutung**
1	Zugriff nur lesend möglich
2	Zugriff nur schreibend möglich
3	Zugriff lesend und schreibend möglich

Für OPC-UA-Zugriffe gibt es die Klasse *OPCUAItem* als Subklasse von *DataItem*. Zwei zusätzliche AutomationML-Attribute (Tabelle 2) beschreiben die Zugriffsmethoden auf den OPC UA Node.

Tabelle 2. AutomationML Attribute von OPCUAItem

AutomationML-Attribut	**Bedeutung**
Identifier	beschreibt durch dessen Wert den Node Identifier und durch dessen *AttributeDataType* die OPC-UA-Codierung zur Identifikation
Namespace	beschreibt den Namensraum in dem sich der OPC UA Node befindet

Das AutomationML-Attribut *Identifier* enthält die Identifikationsmethode des OPC UA Node im *AttributeDataType*-Attribut. Da es mehrere Möglichkeiten gibt, ein OPC UA Node zu identifizieren, werden die Definitionen für den Wert von *AttributeDataType* wie in Tabelle 3 vorgenommen.

Tabelle 3. Mögliche Werte des *AttributeDataType* vom AutomationML-Attribut *Identifier*

Wert von AttributeDataType	**Bedeutung**
xs:string	Identifikation des OPC UA Node über Zeichenkette
xs:integer	Identifikation des OPC UA Node über die Enumeration im OPC UA Server
xs:base64binary	Identifikation des OPC UA Node über Byte Array
xs:ID	Identifikation des OPC UA Node über eine GUID

In der *InstanceList* des *CommunicationSet*-Objekts werden Datenobjekte für jedes Element, das kommuniziert, angelegt. Die Elemente können in verschiedenen Aspekten beschrieben sein. Wollen diese jedoch kommunizieren, so muss für jedes dieser Objekte ein Objekt vom Typ *DataAssembly* – oder ein von diesem abgeleiteter Typ – in der *InstanceList* vorhanden sein.

Die Datenobjekte sind Gegenstand von Blatt 3 dieser Richtlinienreihe, sodass hier im Weiteren lediglich auf die Modellierungsvorschrift eingegangen wird.

Alle *SystemUnitClasses*, die von *DataAssembly* abgeleitet werden, erhalten die zu kommunizierenden Variablen als AutomationML-Attribute. Hierzu haben die abgeleiteten Klassen zum einen bereits vordefinierte Attribute, die von der PFE erwartet werden, lassen sich jedoch durch weitere Attribute erweitern.

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017

6.4.3 Modellierung statischer und dynamischer Zugriffspunkte

Grundsätzlich lassen sich die Zugriffspunkte, die in den *DataAssemblies* modelliert werden, entweder statisch oder dynamisch ausführen.

Bei statischer Ausführung wird der Wert der Modulvariable in den Wert (*Value*) des zugehörigen AutomationML-Attributs des *DataAssemblies* geschrieben. Der *AttributeDataType* des AutomationML-Attributs erhält den entsprechend Wert für den Datentyp der Modulvariable, also z.B. *xs:float* für eine Gleitpunktzahl. Damit ist die Modellierung eines statischen Werts abgeschlossen und kann im MTP somit abgebildet werden.

Dynamische Werte werden ebenfalls innerhalb des AutomationML Attributes mit zugehörigem Namen spezifiziert. Hierbei ist jedoch darauf zu achten, dass der *AttributeDataType* des AutomationML-Attributs zwingend auf *xs:IDREF* zu stellen ist. Das Attribut erhält dann als Wert eine ID. Diese ID entspricht der ID des zugehörigen *DataItem* unterhalb der in der *SourceList* definierten Datenquelle. Hierdurch wird aus der Kommunikationsvariable des *DataAssembly* auf die zugehörige Schnittstelle der Datenquelle des Moduls verwiesen. Diese kann in die PFE integriert werden.

Bild 8 stellt ein Beispiel einer statischen und einer dynamischen Modellierung dar.

6.4.4 Konzept der LinkedObjects

Aus den verschiedenen Aspekten, die im MTP modelliert werden, kann auf den Kommunikationsaspekt, der im Manifest beschrieben ist, verwiesen werden.

Wird z.B. ein Ventil im Bedienbild angezeigt, so soll dies gegebenenfalls die Farbe wechseln, wenn es geschlossen ist. Die hierfür notwendige Variable befindet sich im Manifest, die Beschreibung der Anzeige befindet sich im HMI-Bereich des MTP. Um eine Verbindung zwischen diesen Aspekten zu ermöglichen, werden die SUC *LinkedObject* verwendet.

Die SUC *LinkedObject* befindet sich in der Klassendefinition des Manifests. Die Klasse besitzt lediglich ein Attribut *RefID*, das verwendet wird, um eine Zusammengehörigkeit eines Datenobjekts und eines Objekts eines anderen Aspekts zu beschreiben.

Die Mutterklasse zur Beschreibung der Datenobjekte – *DataAssembly* – ist von *LinkedObject* abgeleitet, wodurch jedes Datenobjekt bereits das AutomationML-Attribut *RefID* besitzt und auf dieses verwiesen werden kann.

Wenn aus einem Aspekt heraus auf die Datenobjekte verwiesen wird, so muss die verweisende Klasse deshalb ebenfalls von *LinkedObject* abgeleitet werden. Hierdurch erhält auch diese *SystemUnitClass* eine *RefID* und kann auf ein anderes *LinkedObject* verweisen.

Um die Zusammengehörigkeit der *LinkedObjects* der verschiedenen Aspekte zu erreichen, werden alle *RefIDs* der zusammengehörigen *LinkedObjects* auf den gleichen Wert bzw. die gleiche ID gesetzt. So ist es möglich,

a) Verbindungen zwischen Aspekten und den Datenobjekten herzustellen,
b) die Zusammengehörigkeit von *InternalElements* verschiedener Aspekte zu modellieren und
c) verschiedene Aspekte auf ein Datenobjekt zu verweisen und damit den gleichen Kommunikationskanal zu benutzen.

Bild 9 zeigt, wie ein Objekt (Symbol) der Visualisierung mit einem Datenobjekt im Manifest verbunden ist. Hierfür erhält das Datenobjekt die gleiche *RefID* wie das Visualisierungsobjekt. Für andere Aspekte wird analog verfahren.

Zusätzlich ist es möglich, mehrere Aspekte mit dem gleichen Datenobjekt zu verbinden. Hierdurch erhält man eine eindeutige Zuordnung zwischen den Aspekten und dem Kommunikationsaspekt.

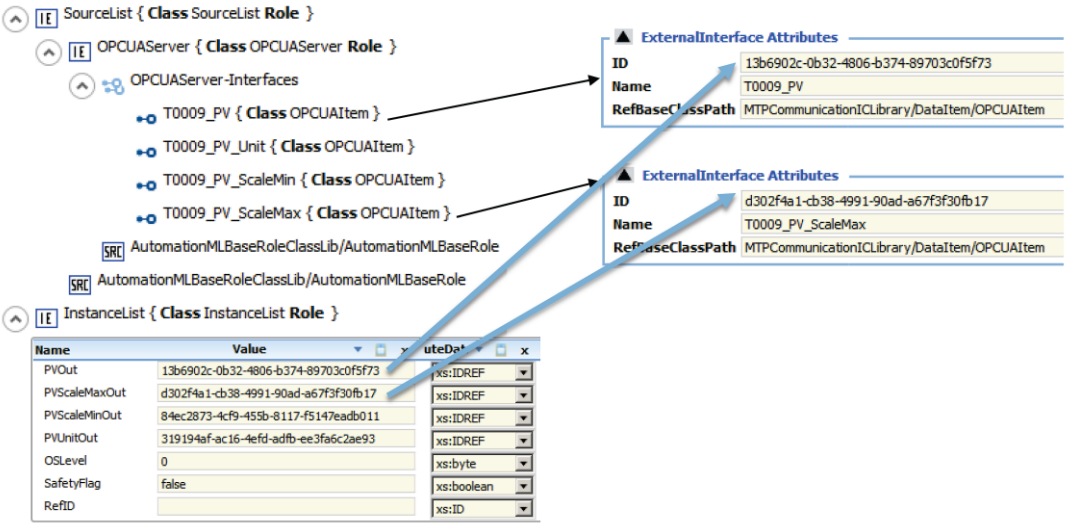


Bild 8. Modellierung der Zugriffspunkte (statisch und dynamisch)

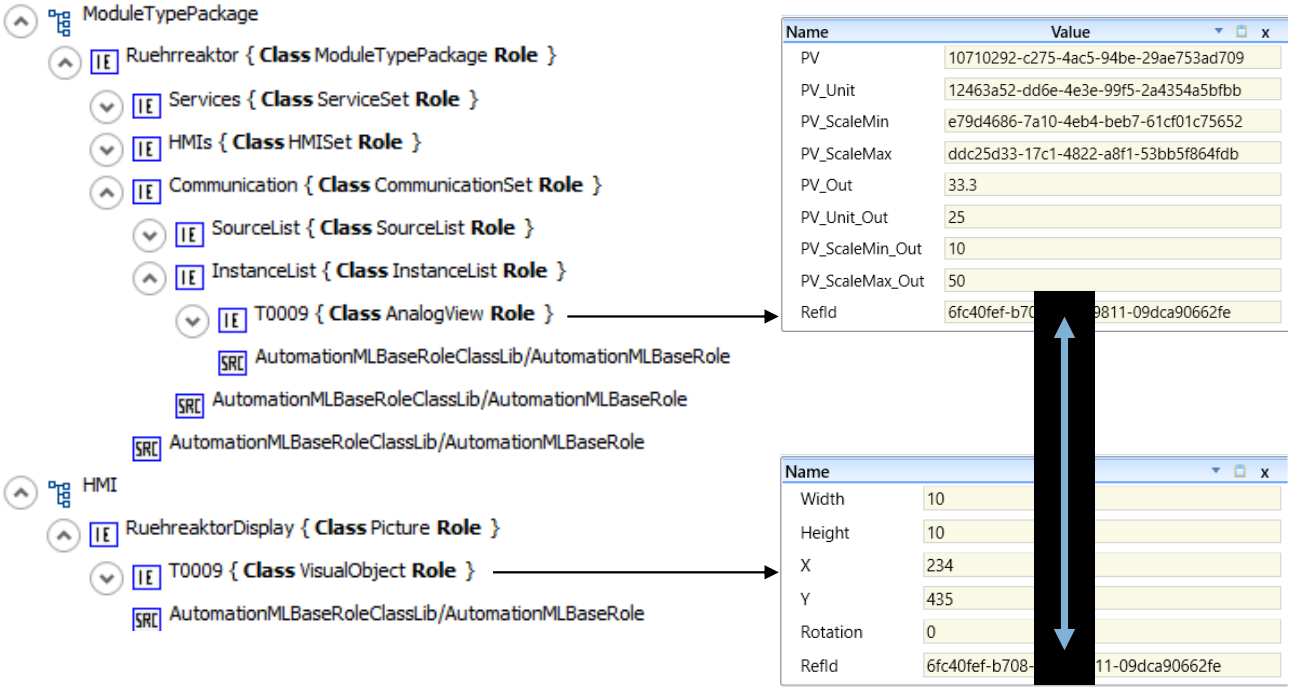


Bild 9. Beispielhafte Verknüpfung von Manifest und HMI über RefId des LinkedObjects

6.4.5 Übersicht der AutomationML-Klassen

Für jeden im MTP zu realisierenden Aspekt wird eine entsprechende *SystemUnitClassLibrary* erstellt, welche die Klassen, die zur Modellierung des Aspekts notwendig sind, beinhaltet. Hieraus ergibt sich eine Bibliotheksstruktur, wie in Bild 6 dargestellt. Werden *ExternalInterfaces* benötigt, so wird für den jeweiligen Aspekt zusätzlich eine *InterfaceClassLibrary* entwickelt.

Die Modellierung des Manifests findet durch Verwendung festgelegter *SystemUnitClasses* (Bild 6) statt. Neben den eigens für das MTP definierten Klassen werden die durch den AutomationML-Verein definierten *InterfaceClasses*, die *AutomationMLInterfaceClassLib*, verwendet.

Die *SystemUnitClasses* werden zur Beschreibung der internen Struktur des Manifests verwendet. Als grundsätzliches Konzept wird im MTP ein Modell-View-Ansatz verwendet, der das Datenmodell eines Objekts von dessen anderen Aspekten wie der Visualisierung oder der Funktionen trennt. Die Datenstrukturen werden immer im Manifest abgebildet, die anderen MTP-Aspekte, z.B. Visualisierung,

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017

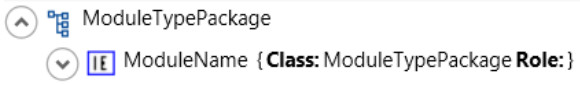
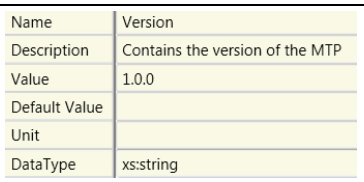
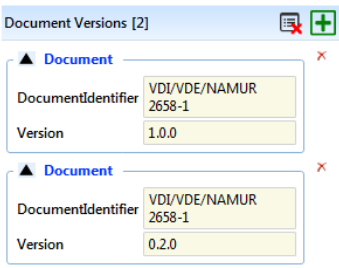
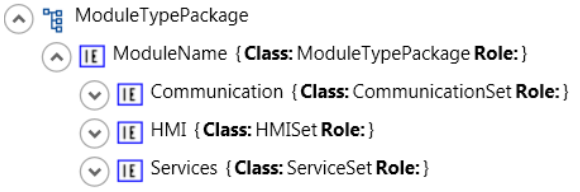
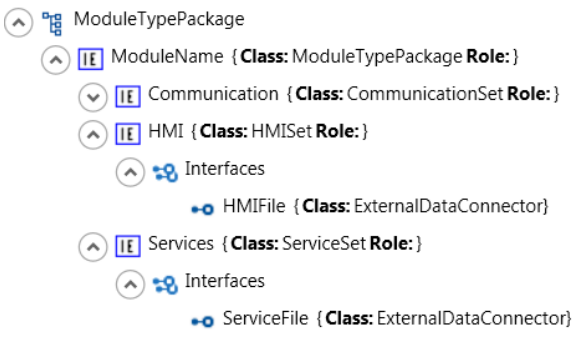
werden in einer gesonderten *InstanceHierarchy* oder einem anderen entsprechenden Modell abgebildet.

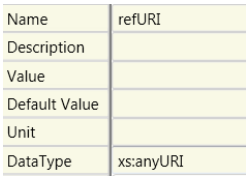
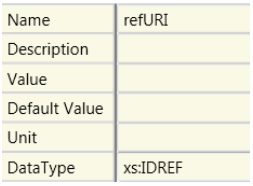
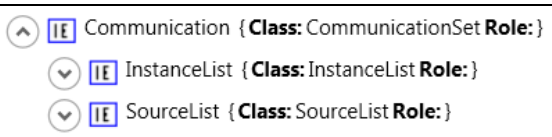
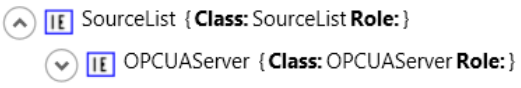
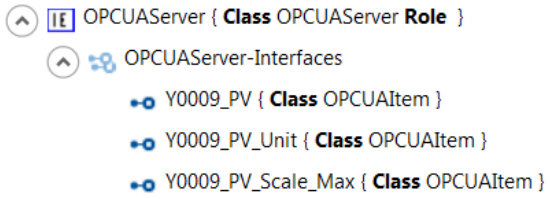
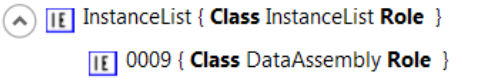
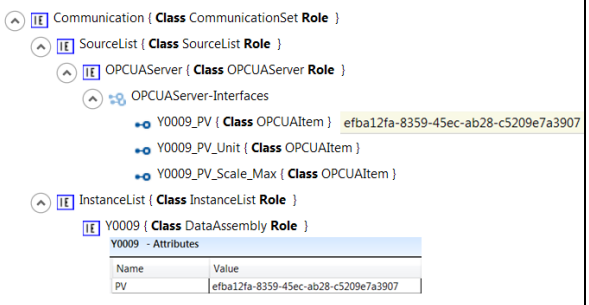
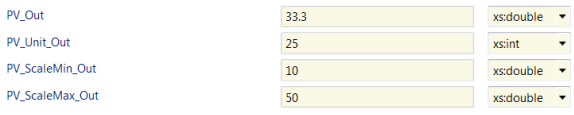
In weiteren Richtlinien dieser Reihe werden zu den Bibliotheken die nötigen Klassen der Aspekte hinzukommen. Für jeden Aspekt, der im MTP abgebildet werden soll, wird eine neue Verzeichnisklasse, abgeleitet von *MTP-Set*, benötigt, und in einer eigenen Bibliothek eingeordnet.

6.5 Modellierungsvorschriften

Tabelle 4 formuliert die in diesem Blatt festgelegte Spezifikation als handlungsorientierte Modellierungsvorschriften, die bei der Erstellung eines MTP eingehalten werden müssen.

Tabelle 4: Modellierungsvorschriften

ID	Vorschrift	Verweis	Beispiel
1	Die *InstanceHierarchy* heißt immer „ModuleTypePackage“	Kap. 6.4	
2	Unter der *InstanceHierarchy* wird ein InternalElement vom Typ *ModuleTypePackage* angelegt, das den Namen des Moduls trägt.	Kap. 6.4	
3	Im *InternalElement* vom Typ *ModuleTypePackage* wird die Version des vom Hersteller generierten MTP durch ein Attribut mit dem Namen *Version* abgebildet. Als *DataType* ist xs:string zu wählen.	Kap. 6.4	
4	Die Version der Richtlinie, die der Generierung der MTP-Instanz und der im MTP enthaltenen Aspekte zugrunde liegt, wird als *AdditionalInformation* gem. [7] angegeben. Als *DocumentIdentifier* ist der jeweilige Name der Richtlinie anzugeben. Das Attribut *Version* enthält die Nummer der Version, die Versionierung erfolgt nach dem Schema Major.Minor.Patch. Die konkreten Werte sind in Abschnitt 3 des jeweiligen Richtlinienblattes definiert.	Kap. 6.4	
5	Im ersten InternalElement des Elements vom Typ *ModuleTypePackage* ist ein InternalElement vom Typ *CommunicationSet* zwingend enthalten. Der Name dieses Elements kann frei gewählt werden.	Kap. 6.4	
6	Zusätzlich können weitere Aspekte durch Hinzufügen von *InternalElements*, deren Typ von *MTPSet* abgeleitet wurde, hinzugefügt werden. Die Namen dieser InternalElements können frei gewählt werden. Im Beispiel rechts *ServiceSet* für den Verweis auf die Dienstebeschreibung und *HMISet* für den Verweis auf die Beschreibungen der Bedienbilder.	Kap. 6.4.1	
7	Das InternalElement dessen Typ von *MTPSet* abgeleitet ist, enthält genau ein *ExternalInterface* des Typs *ExternalDataConnector*.	Kap. 6.4.1	

8	Fallunterscheidung externer Verweis: Verweist das *ExternalInterface* des Typs *ExternalDataConnector* auf ein sich nicht in derselben Datei befindliches Ziel, dann ist der AttributeDataType des Attributes refURI des *ExternalInterface* gleich xs:anyURI zu wählen.	Kap. 6.4.1	
9	Fallunterscheidung interner Verweis: Verweist das *ExternalInterface* des Typs *ExternalDataConnector* auf eine sich in derselben Datei befindliche InstanceHierachy, dann ist der AttributeDataType des Attributes *refURI* des *ExternalInterface* gleich xs:IDREF zu wählen.	Kap. 6.4.1	
10	Das Element vom Typ *CommunicationSet* besitzt immer zwei *InternalElements*: Das im Beispiel „SourceList“ genannte Element ist vom Typ *SourceList* und das „InstanceList“ genannte ist vom Typ *InstanceList*.	Kap. 6.4.2	
11	Die Datenquellen des Moduls werden im *InternalElement* vom Typ *SourceList* abgebildet. Im Beispiel wird hier eine Instanz vom Typ *OPCUAServer* erzeugt.	Kap. 6.4.2	
12	Unterhalb des *InternalElements* der jeweiligen Datenquelle werden die zur Kommunikation zwischen Modul und PFE benötigten Variablen als *ExternalInterface* aufgeführt.	Kap. 6.4.2	
13	Innerhalb des *InternalElements* vom Typ *InstanceList* werden die Variablen der Datenquelle zusätzlich zu Objekten zusammengefasst. Diese Objekte werden als *InternalElements* vom Typ *DataAssembly* oder eine hiervon abgeleitete Klasse modelliert. Diese *InternalElements* enthalten spezifische *Attributes* die auf die Variablen der Datenquelle verweisen. Fehlende *Attributes*, die nicht bereits durch die entsprechende *SystemUnitClass* spezifiziert sind, können bei Bedarf hinzugefügt werden.	Kap. 6.4.2	
14	Die Beziehung zwischen Variablen der Datenquellen und einem entsprechenden Element unterhalb des *InternalElements* vom Typ *InstanceList* wird durch einen Verweis hergestellt. Hierzu erhalten die *Attributes* des *DataAssembly* den *AttributeDataType* xs:IDREF und als Wert eine ID, die der ID des zugehörigen *Externalinterface* unterhalb des *InternalElements* vom Typ *SourceList* entspricht.	Kap. 6.4.3	
15	Konstanten werden als *Attributes* der *InternalElements* unterhalb des *InternalElements* vom Typ *InstanceList* modelliert. Hierzu werden der Value und der *AttributeDataType* des zugehörigen *Attributes* angegeben.	Kap. 6.4.3	

6.6 Beispiel eines Manifests

Bild 10 stellt ein in AutomationML modelliertes MTP dar. Die Serialisierung des Beispiels in XML ist im Anhang A zu sehen. Die Struktur des Manifests sieht für jedes MTP gleich aus.

Beschrieben wird ein Link zur Dienstschnittstellenbeschreibung und ein Link zur Beschreibung eines Bedienbilds. Der Bereich *Communication* enthält die Information zu den verwendeten Kommunikationsmechanismen, in Bild 10 eine OPC-UA-Schnittstelle. Hierzu gehört sowohl der Zugriffspunkt auf den OPC UA Server selbst als auch die verwendeten

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017

OPC UA Nodes. Die OPC UA Nodes gehören zu den Datenobjekten, welche die Kommunikationsschnittstelle des Anlagenequipments abbilden.

Es wird für jeden zu realisierenden Dienst und für jedes anzuzeigende Element ein Datenobjekt benötigt. In Bild 10 wird exemplarisch ein Datenobjekt für das Symbol *T0009* abgebildet.

ExternalInterfaces vom Typ *OPCUAItem* werden verwendet, um für jedes AutomationML-Attribut der Datenklasse die Zugriffsmöglichkeiten auf dem zugehörigen OPC UA Server zu beschreiben.

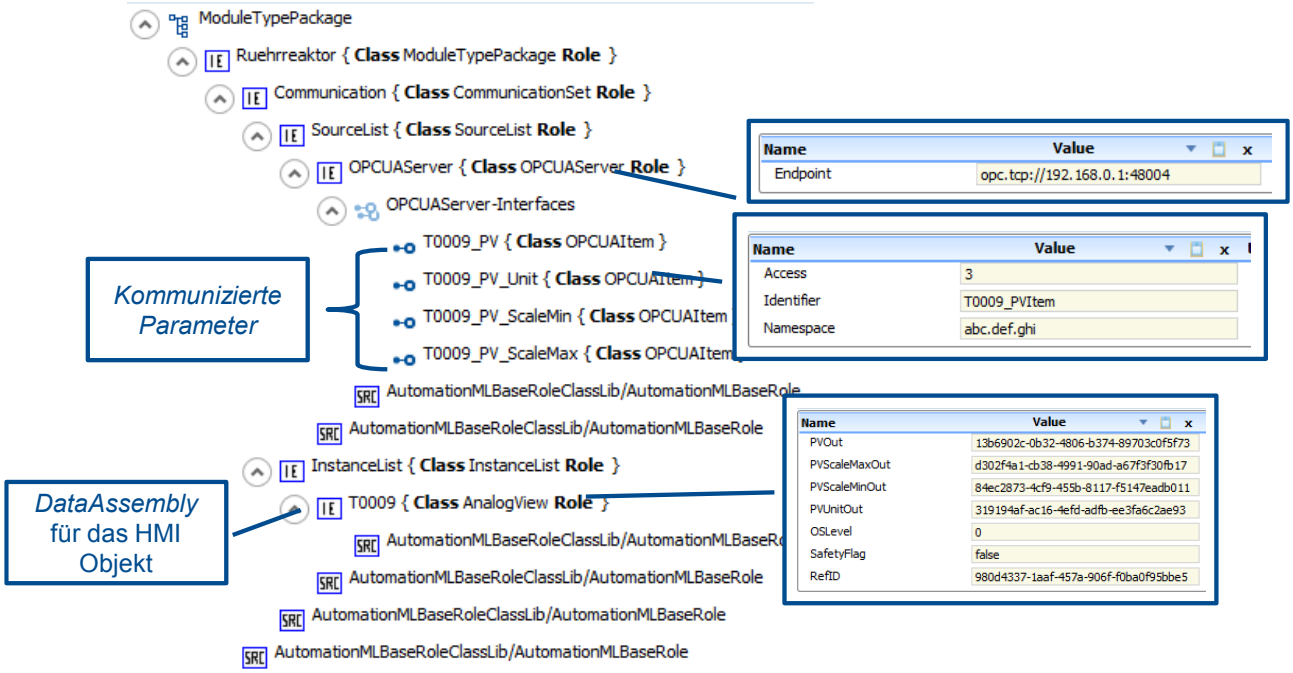


Bild 10. Beispiel eines Manifests und interner Aufbau

Anhang A Beispiel eines Manifestes in AutomationML mit zugehörigen Klassen MTPSUCLib, MTPDataObjectSUCLib, MTPCommunicationSUCLib, MTPCommunicationICLib

```
<?xml version="1.0" encoding="utf-8"?>
<CAEXFile FileName="Richtlinie Blatt 1 - Beispiel AML und Klassen.aml" SchemaVersion="2.15" xsi:noNamespaceSchemaLocation="CAEX_ClassModel_V2.15.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Version>2.0</Version>
  <Copyright>VDI/VDE GMA 5.16 (c) 2016</Copyright>
  <AdditionalInformation DocumentVersions="Recommendations">
    <Document DocumentIdentifier="VDI/VDE/NAMUR 2658-1" Version="1.0.0" />
  </AdditionalInformation>
  <AdditionalInformation>
    <WriterHeader>
      <WriterName>GMA 5.16</WriterName>
      <WriterID>VDI/VDE</WriterID>
      <WriterVendor>VDI/VDE GMA 5.16 - Zukünftige Architekturen der Automatisierungstechnik</WriterVendor>
      <WriterVendorURL>www.vdi.de</WriterVendorURL>
      <WriterVersion>1.0</WriterVersion>
      <WriterRelease>1.0.0_0.0.0</WriterRelease>
      <LastWritingDateTime>2016-12-20T14:21:01+01:00</LastWritingDateTime>
      <WriterProjectTitle>Module Type Package Libraries for AutomationML</WriterProjectTitle>
      <WriterProjectID>VDI/VDE 2568</WriterProjectID>
    </WriterHeader>
  </AdditionalInformation>
  <AdditionalInformation AutomationMLVersion="2.0" />
  <InstanceHierarchy Name="ModuleTypePackage">
    <Version>1.0.0</Version>
    <InternalElement Name="Ruehrreaktor" RefBaseSystemUnitPath="MTPSUCLib/ModuleTypePackage" ID="e5a1829e-ff39-450d-8838-b513fe33f214">
      <Attribute Name="Version" AttributeDataType="xs:string">
        <Description>Contains the Version of the MTP</Description>
        <DefaultValue>1.0.0</DefaultValue>
        <Value>1.0.0</Value>
      </Attribute>
      <InternalElement Name="Communication" RefBaseSystemUnitPath="MTPSUCLib/CommunicationSet" ID="5d0c8328-2a34-496d-89ed-179f93464890">
        <InternalElement Name="SourceList" RefBaseSystemUnitPath="MTPSUCLib/CommunicationSet/SourceList" ID="ec9ea048-166e-4958-ad83-da5d334f55dd">
          <InternalElement Name="OPCUAServer" RefBaseSystemUnitPath="MTPCommunicationSUCLib/ServerAssembly/OPCUAServer" ID="0d442235-84e5-4f53-817b-16730ea352cf">
            <Attribute Name="Endpoint" Unit="" AttributeDataType="xs:string">
              <Description>Endpoint URL of the server - like 'opc.tcp://192.186.2.1:5555</Description>
              <DefaultValue />
              <Value>opc.tcp://192.168.0.1:48004</Value>
            </Attribute>
            <ExternalInterface Name="T0009_PV" RefBaseClassPath="MTPCommunicationICLib/DataItem/OPCUAItem" ID="13b6902c-0b32-4806-b374-89703c0f5f73">
              <Attribute Name="Access" Unit="" AttributeDataType="xs:unsignedByte">
                <Description>Bit Definition: [0] = NoAccess ; [1] = Read ; [2] = Write ; [3] = Read/Write</Description>
                <DefaultValue>0</DefaultValue>
                <Value>3</Value>
              </Attribute>
              <Attribute Name="Identifier" Unit="" AttributeDataType="xs:string">
                <Description>Identifier of the OPC UA Node - IdentifierType (depends on AttributeDataType): xs:string = string ; xs:ID = GUID ; xs:Base64Binary = ByteArray ; xs:int = Integer</Description>
                <DefaultValue />
                <Value>T0009_PVItem</Value>
              </Attribute>
              <Attribute Name="Namespace" Unit="" AttributeDataType="xs:string">
                <Description>Namespace of the OPC UA Node</Description>
                <DefaultValue />
                <Value>abc.def.ghi</Value>
              </Attribute>
            </ExternalInterface>
            <ExternalInterface Name="T0009_PV_Unit" RefBaseClassPath="MTPCommunicationICLib/DataItem/OPCUAItem" ID="319194af-ac16-4efd-adfb-ee3fa6c2ae93">
              <Attribute Name="Access" Unit="" AttributeDataType="xs:unsignedByte">
                <Description>Bit Definition: [0] = NoAccess ; [1] = Read ; [2] = Write ; [3] = Read/Write</Description>
                <DefaultValue>0</DefaultValue>
                <Value />
              </Attribute>
              <Attribute Name="Identifier" Unit="" AttributeDataType="xs:string">
                <Description>Identifier of the OPC UA Node - IdentifierType (depends on AttributeDataType): xs:string = string ; xs:ID = GUID ; xs:Base64Binary = ByteArray ; xs:int = Integer</Description>
```

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017


```
                        <DefaultValue />
                        <Value />
                    </Attribute>
                    <Attribute Name="Namespace" Unit="" AttributeDataType="xs:string">
                        <Description>Namespace of the OPC UA Node</Description>
                        <DefaultValue />
                        <Value />
                    </Attribute>
                </ExternalInterface>
                <ExternalInterface Name="T0009_PV_ScaleMin" RefBaseClassPath="MTPCommunicationICLib/DataItem/OPCUAItem"
ID="84ec2873-4cf9-455b-8117-f5147eadb011">
                    <Attribute Name="Access" Unit="" AttributeDataType="xs:unsignedByte">
                        <Description>Bit Definition: [0] = NoAccess ; [1] = Read ; [2] = Write ; [3] = Read/Write</Description>
                        <DefaultValue>0</DefaultValue>
                        <Value />
                    </Attribute>
                    <Attribute Name="Identifier" Unit="" AttributeDataType="xs:string">
                        <Description>Identifier of the OPC UA Node - IdentifierType (depends on AttributeDataType): xs:string = string ;
xs:ID = GUID ; xs:Base64Binary = ByteArray ; xs:int = Integer</Description>
                        <DefaultValue />
                        <Value />
                    </Attribute>
                    <Attribute Name="Namespace" Unit="" AttributeDataType="xs:string">
                        <Description>Namespace of the OPC UA Node</Description>
                        <DefaultValue />
                        <Value />
                    </Attribute>
                </ExternalInterface>
                <ExternalInterface Name="T0009_PV_ScaleMax" RefBaseClassPath="MTPCommunicationICLib/DataItem/OPCUAItem"
ID="d302f4a1-cb38-4991-90ad-a67f3f30fb17">
                    <Attribute Name="Access" Unit="" AttributeDataType="xs:unsignedByte">
                        <Description>Bit Definition: [0] = NoAccess ; [1] = Read ; [2] = Write ; [3] = Read/Write</Description>
                        <DefaultValue>0</DefaultValue>
                        <Value />
                    </Attribute>
                    <Attribute Name="Identifier" Unit="" AttributeDataType="xs:string">
                        <Description>Identifier of the OPC UA Node - IdentifierType (depends on AttributeDataType): xs:string = string ;
xs:ID = GUID ; xs:Base64Binary = ByteArray ; xs:int = Integer</Description>
                        <DefaultValue />
                        <Value />
                    </Attribute>
                    <Attribute Name="Namespace" Unit="" AttributeDataType="xs:string">
                        <Description>Namespace of the OPC UA Node</Description>
                        <DefaultValue />
                        <Value />
                    </Attribute>
                </ExternalInterface>
                <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
            </InternalElement>
            <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
        </InternalElement>
        <InternalElement Name="InstanceList" RefBaseSystemUnitPath="MTPSUCLib/CommunicationSet/InstanceList" ID="46e1fbce-68a9-
4304-b28c-e0616d7a3ed5">
            <InternalElement Name="T0009" RefBaseSystemUnitPath="MTPDataObjectSUCLib/DataAssembly/AnalogView" ID="ae9f0b7d-
4952-40e5-b96b-2b3c0e4db0c0">
                <Attribute Name="PVOut" Unit="" AttributeDataType="xs:IDREF">
                    <Description>AnalogValue - Formats: xs:double = REAL; xs:int = INT; xs:unsignedInt = WORD; xs:unsignedLong = DWORD;
xs:byte = BYTE; xs:long = DINT</Description>
                    <DefaultValue />
                    <Value>13b6902c-0b32-4806-b374-89703c0f5f73</Value>
                    <RefSemantic CorrespondingAttributePath="Connectable" />
                    <RefSemantic CorrespondingAttributePath="Standard" />
                </Attribute>
                <Attribute Name="PVScaleMaxOut" Unit="" AttributeDataType="xs:IDREF">
                    <Description>ScaleMax - Formats: xs:double = REAL; xs:int = INT; xs:unsignedInt = WORD; xs:unsignedLong = DWORD;
xs:byte = BYTE; xs:long = DINT</Description>
                    <DefaultValue />
                    <Value>d302f4a1-cb38-4991-90ad-a67f3f30fb17</Value>
                    <RefSemantic CorrespondingAttributePath="Connectable" />
                    <RefSemantic CorrespondingAttributePath="Standard" />
                </Attribute>
                <Attribute Name="PVScaleMinOut" Unit="" AttributeDataType="xs:IDREF">
                    <Description>ScaleMin - Formats: xs:double = REAL; xs:int = INT; xs:unsignedInt = WORD; xs:unsignedLong = DWORD;
xs:byte = BYTE; xs:long = DINT</Description>
                    <DefaultValue />
                    <Value>84ec2873-4cf9-455b-8117-f5147eadb011</Value>
                    <RefSemantic CorrespondingAttributePath="Connectable" />
```

```
                        <RefSemantic CorrespondingAttributePath="Standard" />
                    </Attribute>
                    <Attribute Name="PVUnitOut" Unit="" AttributeDataType="xs:IDREF">
                        <Description>Unit</Description>
                        <DefaultValue />
                        <Value>319194af-ac16-4efd-adfb-ee3fa6c2ae93</Value>
                        <RefSemantic CorrespondingAttributePath="Connectable" />
                        <RefSemantic CorrespondingAttributePath="Standard" />
                    </Attribute>
                    <Attribute Name="OSLevel" Unit="" AttributeDataType="xs:byte">
                        <Description>OSLevel which has the actual clearance to write into the plc (required if multiple OS Stations are available (local,
central, remote)</Description>
                        <DefaultValue>0</DefaultValue>
                        <Value>0</Value>
                    </Attribute>
                    <Attribute Name="SafetyFlag" Unit="" AttributeDataType="xs:boolean">
                        <Description>The Safety Flag signs that this data structure contains variables out of a safety relevant system (only reading) or
not</Description>
                        <DefaultValue>false</DefaultValue>
                        <Value>false</Value>
                    </Attribute>
                    <Attribute Name="RefID" Unit="" AttributeDataType="xs:ID">
                        <Description>Reference ID - Relation of different data sets to a complete described Instance Object</Description>
                        <DefaultValue />
                        <Value>980d4337-1aaf-457a-906f-f0ba0f95bbe5</Value>
                    </Attribute>
                    <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
                </InternalElement>
                <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
            </InternalElement>
            <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
        </InternalElement>
        <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
    </InternalElement>
</InstanceHierarchy>
<InterfaceClassLib Name="AutomationMLInterfaceClassLib">
    <Description>Standard Automation Markup Language Interface Class Library</Description>
    <Version>2.2.0</Version>
    <InterfaceClass Name="AutomationMLBaseInterface">
        <InterfaceClass Name="Order" RefBaseClassPath="AutomationMLBaseInterface">
            <Attribute Name="Direction" AttributeDataType="xs:string" />
        </InterfaceClass>
        <InterfaceClass Name="PortConnector" RefBaseClassPath="AutomationMLBaseInterface" />
        <InterfaceClass Name="InterlockingConnector" RefBaseClassPath="AutomationMLBaseInterface" />
        <InterfaceClass Name="PPRConnector" RefBaseClassPath="AutomationMLBaseInterface" />
        <InterfaceClass Name="ExternalDataConnector" RefBaseClassPath="AutomationMLBaseInterface">
            <Attribute Name="refURI" AttributeDataType="xs:anyURI" />
            <InterfaceClass Name="COLLADAInterface" RefBaseClassPath="ExternalDataConnector" />
            <InterfaceClass Name="PLCopenXMLInterface" RefBaseClassPath="ExternalDataConnector" />
        </InterfaceClass>
        <InterfaceClass Name="Communication" RefBaseClassPath="AutomationMLBaseInterface">
            <InterfaceClass Name="SignalInterface" RefBaseClassPath="Communication" />
        </InterfaceClass>
    </InterfaceClass>
</InterfaceClassLib>
<InterfaceClassLib Name="MTPCommunicationICLib">
    <Description>InterfaceClassLibrary who contains the base interfaces of a MTP Communication Description</Description>
    <InterfaceClass Name="DataItem">
        <Description>InterfaceClass who represents the a common source item - all defined source items derive from SourceItem</Description>
        <Attribute Name="Access" Unit="" AttributeDataType="xs:unsignedByte">
            <Description>Bit Definition: [0] = NoAccess ; [1] = Read ; [2] = Write ; [3] = Read/Write</Description>
            <DefaultValue>0</DefaultValue>
            <Value />
        </Attribute>
        <InterfaceClass Name="OPCUAItem" RefBaseClassPath="MTPCommunicationICLib/DataItem">
            <Description>Base class for a Profinet data item - not specified only an example</Description>
            <Attribute Name="Identifier" Unit="" AttributeDataType="xs:string">
                <Description>Identifier of the OPC UA Node - IdentifierType (depends on AttributeDataType): xs:string = string ; xs:ID = GUID ;
xs:Base64Binary = ByteArray ; xs:int = Integer</Description>
                <DefaultValue />
                <Value />
            </Attribute>
            <Attribute Name="Namespace" Unit="" AttributeDataType="xs:string">
                <Description>Namespace of the OPC UA Node</Description>
                <DefaultValue />
                <Value />
            </Attribute>
```

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017


```
        </InterfaceClass>
      </InterfaceClass>
    </InterfaceClassLib>
    <RoleClassLib Name="AutomationMLBaseRoleClassLib">
      <Description>Automation Markup Language base role class library</Description>
      <Version>2.2.0</Version>
      <RoleClass Name="AutomationMLBaseRole">
        <RoleClass Name="Group" RefBaseClassPath="AutomationMLBaseRole">
            <Attribute Name="AssociatedFacet" AttributeDataType="xs:string" />
        </RoleClass>
        <RoleClass Name="Facet" RefBaseClassPath="AutomationMLBaseRole" />
        <RoleClass Name="Port" RefBaseClassPath="AutomationMLBaseRole">
            <Attribute Name="Direction" AttributeDataType="xs:string" />
            <Attribute Name="Cardinality">
              <Attribute Name="MinOccur" AttributeDataType="xs:unsignedInt" />
              <Attribute Name="MaxOccur" AttributeDataType="xs:unsignedInt" />
            </Attribute>
            <Attribute Name="Category" AttributeDataType="xs:string" />
            <ExternalInterface Name="ConnectionPoint" ID="9942bd9c-c19d-44e4-a197-11b9edf264e7" RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/PortConnector" />
        </RoleClass>
        <RoleClass Name="Resource" RefBaseClassPath="AutomationMLBaseRole" />
        <RoleClass Name="Product" RefBaseClassPath="AutomationMLBaseRole" />
        <RoleClass Name="Process" RefBaseClassPath="AutomationMLBaseRole" />
        <RoleClass Name="Structure" RefBaseClassPath="AutomationMLBaseRole">
            <RoleClass Name="ProductStructure" RefBaseClassPath="Structure" />
            <RoleClass Name="ProcessStructure" RefBaseClassPath="Structure" />
            <RoleClass Name="ResourceStructure" RefBaseClassPath="Structure" />
        </RoleClass>
        <RoleClass Name="PropertySet" RefBaseClassPath="AutomationMLBaseRole" />
      </RoleClass>
    </RoleClassLib>
    <SystemUnitClassLib Name="MTPSUCLib">
      <Description>SystemUnitClassLibrary who contains the base elements of a MTP Package</Description>
      <SystemUnitClass Name="MTPSet">
        <Description>SystemUnitClass who represents the entry point into a MTP Package</Description>
        <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
      </SystemUnitClass>
      <SystemUnitClass Name="LinkedObject">
        <Description>Base Class for CrossSet referencing of ObjectInstances</Description>
        <Attribute Name="RefID" Unit="" AttributeDataType="xs:ID">
            <Description>Reference ID - Relation of different data sets to a complete described Instance Object</Description>
            <DefaultValue />
            <Value />
        </Attribute>
        <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
      </SystemUnitClass>
      <SystemUnitClass Name="ModuleTypePackage">
        <Attribute Name="Version" AttributeDataType="xs:string">
            <Description>Contains the Version of the MTP</Description>
            <DefaultValue>1.0.0</DefaultValue>
            <Value>1.0.0</Value>
        </Attribute>
        <InternalElement Name="CommunicationSet" RefBaseSystemUnitPath="MTPSUCLib/CommunicationSet" ID="438d7919-2bd5-483c-b993-64153d6b3081">
            <InternalElement Name="InstanceList" RefBaseSystemUnitPath="MTPSUCLib/CommunicationSet/InstanceList" ID="3b0fd67c-4704-4908-803d-cff34f6134d2">
              <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
            </InternalElement>
            <InternalElement Name="SourceList" RefBaseSystemUnitPath="MTPSUCLib/CommunicationSet/SourceList" ID="7e31053b-3ab3-4822-bbcf-cfaee8f47452">
              <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
            </InternalElement>
            <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
        </InternalElement>
        <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
      </SystemUnitClass>
      <SystemUnitClass Name="CommunicationSet" RefBaseClassPath="MTPSUCLib/MTPSet">
        <Description>Base class for the MTP Communication Facet </Description>
        <InternalElement Name="InstanceList" RefBaseSystemUnitPath="MTPSUCLib/CommunicationSet/InstanceList" ID="77f97d39-817d-46f8-9a83-9412e096c9d1">
            <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
        </InternalElement>
        <InternalElement Name="SourceList" RefBaseSystemUnitPath="MTPSUCLib/CommunicationSet/SourceList" ID="7f00acbe-07f5-46d5-a98e-1bd584f82697">
            <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
        </InternalElement>
```

```
        <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
        <SystemUnitClass Name="InstanceList">
            <Description>SystemUnitClass for the List of DataAssemblies</Description>
            <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
        </SystemUnitClass>
        <SystemUnitClass Name="SourceList">
            <Description>SystemUnitClass for the List of Server Sources</Description>
            <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
        </SystemUnitClass>
    </SystemUnitClass>
  </SystemUnitClassLib>
  <SystemUnitClassLib Name="MTPCommunicationSUCLib">
    <Description>SystemUnitClassLibrary who contains the base elements of a MTP Communication Description</Description>
    <SystemUnitClass Name="ServerAssembly">
      <Description>SystemUnitClass who represents the a common source object - all defined sources derive from SourceAssembly</Description>
      <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
      <SystemUnitClass Name="OPCUAServer" RefBaseClassPath="MTPCommunicationSUCLib/ServerAssembly">
          <Description>Base class for all opc ua server instances inside the communication facets</Description>
          <Attribute Name="Endpoint" Unit="" AttributeDataType="xs:string">
            <Description>Endpoint URL of the server - like 'opc.tcp://192.186.2.1:5555</Description>
            <DefaultValue />
            <Value />
          </Attribute>
          <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
      </SystemUnitClass>
    </SystemUnitClass>
  </SystemUnitClassLib>
  <SystemUnitClassLib Name="MTPDataObjectSUCLib">
    <Description>SystemUnitClassLibrary who contains the base elements of a MTP DataAssemblies</Description>
    <SystemUnitClass Name="DataAssembly" RefBaseClassPath="MTPSUCLib/LinkedObject">
      <Description> </Description>
      <Attribute Name="OSLevel" Unit="" AttributeDataType="xs:byte">
          <Description>OSLevel which has the actual clearance to write into the plc (required if multiple OS Stations are available (local, central,
remote)</Description>
          <DefaultValue>0</DefaultValue>
          <Value />
      </Attribute>
      <SupportedRoleClass RefRoleClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
    </SystemUnitClass>
  </SystemUnitClassLib>
</CAEXFile>
```

 Alle Rechte vorbehalten © Verein Deutscher Ingenieure e.V., Düsseldorf 2017

Schrifttum

Technische Regeln

DIN EN 61508*VDE 0803 Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme. Berlin: Beuth Verlag

DIN EN 61511*VDE 0810 Funktionale Sicherheit; Sicherheitstechnische Systeme für die Prozessindustrie. Berlin: Beuth Verlag

Namur NE 148:2013-10 Anforderungen an die Automatisierungstechnik durch die Modularisierung verfahrenstechnischer Anlagen. Leverkusen: NAMUR – Interessengemeinschaft Automatisierungstechnik der Prozessindustrie e.V.

SN EN 61131-3*IEC 61131-3:2013 Speicherprogrammierbare Steuerungen; Teil 3: Programmiersprachen. Berlin: Beuth Verlag

SN EN 62714*IEC 62714 Datenaustauschformat für Planungsdaten industrieller Automatisierungssysteme; Automation markup language. Berlin: Beuth Verlag

VDI 1000:2017-02 VDI-Richtlinienarbeit; Grundsätze und Anleitungen. Berlin: Beuth Verlag

VDI/VDE 2180 Sicherung von Anlagen der Verfahrenstechnik mit Mitteln der Prozessleittechnik (PLT). Berlin: Beuth Verlag

VDI/VDE 2182 Informationssicherheit in der industriellen Automatisierung. Berlin: Beuth Verlag

VDI/VDE 2658 Blatt 2 Automatisierungstechnisches Engineering modularer Anlagen in der Prozessindustrie; Modellierung von Bedienbildern (in Vorbereitung)

VDI/VDE 2658 Blatt 3 Automatisierungstechnisches Engineering modularer Anlagen in der Prozessindustrie; Bibliothek für Datenobjekte (in Vorbereitung)

VDI/VDE 2658 Blatt 4 Automatisierungstechnisches Engineering modularer Anlagen in der Prozessindustrie; Modellierung von Moduldiensten (in Vorbereitung)

VDI/VDE 2658 Blatt 5 Automatisierungstechnisches Engineering modularer Anlagen in der Prozessindustrie; Laufzeit und Kommunikationsaspekte (in Vorbereitung)

Weiterführende technische Regeln

DIN EN 61512-2:2003-10 Chargenorientierte Fahrweise; Teil 2: Datenstrukturen und Leitfaden für Sprachen (IEC 61512-2: 2001); Deutsche Fassung EN 61512-2:2002. Berlin: Beuth Verlag

ISA 106:2013-01 Procedure Automation Control for Continuous Process Operations; Models and Terminology. North Carolina: International Society of Automation

Literatur

[1] *Urbas*, *L.*; *Bleuel*, *S.* et al: Automatisierung von Prozessmodulen; Von Package-Unit-Integration zu modularen Anlagen. atp edition 54 (2012) 1-2, S. 44–52

[2] *Obst*, *M.*; *Holm*, *T.* et al: Automatisierung im Life Cycle modularer Anlagen; Welche Veränderungen und Chancen ergeben sich. atp edition 55 (2013) 1-2, S. 24–31

[3] *Hoernicke*, *M.*; *Holm*, *T.* et al.: Technologiebewertung zur Beschreibung für verfahrenstechnische Module – Ergebnisse des Namur AK 1.12.1. In: Kongress *Automation 2016*, Baden-Baden

[4] *Hoernicke*, *M.*; *Messinger*, *C.*; *Arroyo*, *E.*; *Fay*, *A.*: Topologiemodelle in AutomationML; Objektorientierte Grundlage für die Automatisierung der Automatisierung. atp edition 58 (2016) 5, S. 28–41

[5] *Bernshausen*, *J.*; *Haller*, *A.* et al.: Namur Module Type Package – Definition. atp edition 58 (2016) 1-2, S. 72–81

[6] Temporärer ProcessNet-Arbeitskreis Modulare Anlagen (Ed.). (2016). Modular Plants: Flexible chemical production by modularization and standardization – status quo and future trends. Retrieved from http://process-net.org/dechema_media/ModularPlants_2016-p-20002425.pdf

[7] AutomationML (2016). Best Practice Recommendations: Naming of related Documents and their Versions. BPR RefVersion V1.0.0. Retrieved from https://www.automationml.org/o.red/uploads/dateien/1481372636-BPR_006E_Naming_Documents_and_Versions_Dec2016.pdf