

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.

An der Erarbeitung dieser Richtlinie waren beteiligt:

Henry Bloch, Hamburg
Mario Hoernicke, Ladenburg
Katharina Stark, Ladenburg
Thomas Holm, Minden
Mathias Maurmaier, Karlsruhe
Andreas Stutz, Karlsruhe
Leon Urbas, Dresden (Vorsitz)
Stephan Hensel, Dresden
Polyana da Silva Santos, Marl
Alexander Kehl, Esslingen
Andre Pomraenke, Magdeburg
Christian Roth, Offenburg
Stefan Scheffler, Offenburg
Alfons Fehrenbacher, Offenburg
Christian Schäfer, Darmstadt
Oleg Makarov, Mannheim
Anselm Klose, Dresden
Simon Löpker, Minden

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. 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 der vorliegenden Richtlinie werden folgende Aspekte fokussiert:

- Konzept der Dienstschnittstelle von Modulen
- Zustands- und Dienstmodelle modularer Anlagen
- Strukturierung der Dienstschnittstelle des MTP
- Schnittstellen des Verhaltensvertrags zwischen Diensten des Moduls und der PFE
- Definition und Beschreibung des Dienstschnittstellenmodells
- Definition und Beschreibung der Dienstparameter
- Definition und Beschreibung von Fahrweisen
- Definition und Beschreibung der Verriegelung von Diensten
- Modellierungsvorschriften zur Erstellung der Dienstschnittstelle

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

- Blatt 1: Allgemeines Konzept und Schnittstellen
- Blatt 2: Modellierung von Bedienbildern
- Blatt 3: Bibliothek für Datenobjekte
- Blatt 5: Laufzeit- und Kommunikationsaspekte
- Blatt 6: Alarmmanagement

Zusätzlich geplant sind Richtlinien zu den Themen „Diagnose“, „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

Diese Richtlinie definiert die Modellierungsvorschriften von Moduldiensten für prozesstechnische Module gemäß [VDI/VDE/NAMUR 2658 Blatt 1]. Zielgruppen sind die gleichen wie in [VDI/VDE/NAMUR 2658 Blatt 1]. Anwendungsfälle und Definitionen sind die gleichen wie bereits in [VDI/VDE/NAMUR 2658 Blatt 1] definiert.

2 Dienste

2.1 Grundkonzept dienstbasierter Prozessführung modularer Anlagen

Die in den Modulen vorgesehenen prozesstechnischen Funktionen werden als Dienste gekapselt. Ein Reaktormodul mit einem Rührer beispielsweise, könnte demnach 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 in z.B. „BefüllenA“ und „BefüllenB“ unterscheiden kann. Verfügt der Reaktor über ein Heizsystem, kann ebenfalls der Dienst „Heizen“ ausgeführt werden. Ein entsprechender Parametersatz dieses Dienstes könnte die Zieltemperatur, die Temperatur-Anstiegsgeschwindigkeit und die Haltezeit sein.

Die Funktion, die Dienste der einzelnen Module in eine für die Produktion des gewünschten Produktes erforderliche Folge zu bringen (Dienste-Orchestrierung), muss von einer noch während des Integrations-Engineering gestaltbaren Automatisierungsinstanz übernommen werden. Diese zusätzliche Orchestrierungsfunktion wird erst durch Kombination verschiedener Module notwendig und kann daher nicht von den einzelnen Modulen übernommen werden. So muss zum Beispiel bei einem kontinuierlich betriebenen Reaktionsprozess das Anfahren des Reaktors mit dem Vorlegen der Ausgangsprodukte abgestimmt werden.

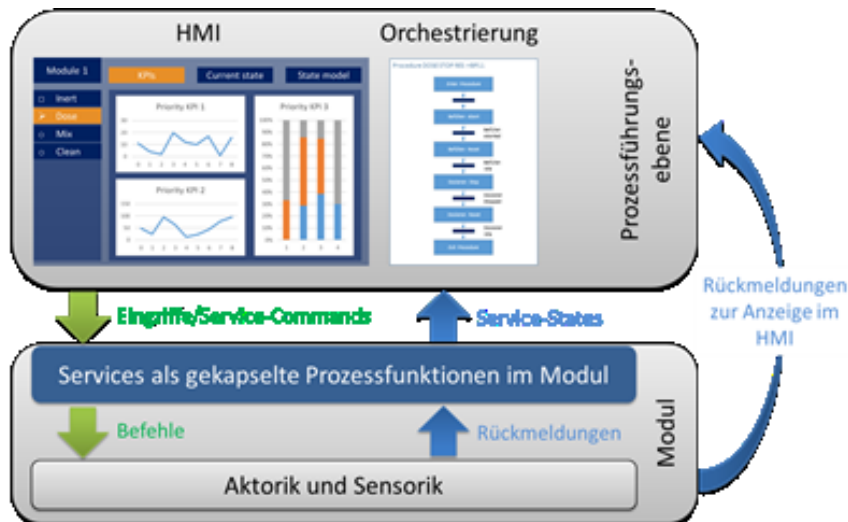


Abbildung 1: Grundlegende Kommunikation zur dienstbasierten Prozessführung [H. Bloch, S. Hensel, M. Hoernicke, K. Stark, A. Menschner, L. Urbas, A. Fay, T. Knohl, J. Bernshausen, A. Haller: Zustandsbasierte Prozessführung modularer Prozessanlagen. Atp edition 11/2017]

Die Dienste werden zur Einflussnahme auf die Module bei der dienstbasierten Prozessführung verwendet. Hierbei folgen die Dienste ähnlich dem Konzept der [ISA 88] bzw. der [IEC 61512] einem fest definierten und nicht konfigurierbaren Zustandsautomaten. Die Dienste-Orchestrierung sendet einen Befehl zum Zustandswechsel eines Dienstes an das Modul. Innerhalb eines jeden Zustandes werden verschiedene Programme, z.B. Abläufe nach [IEC 61131-3], ausgeführt. Die Programme innerhalb der Zustände steuern dann die für die Funktion notwendigen Feldgeräte an und werten die entsprechenden Signale der Sensoren aus. Verschiedene Zustände dürfen die gleiche Funktionalität beinhalten. So kann das Stoppen (im Zustand STOPPING) und das Abbrechen (im Zustand ABORTING) eines vergleichsweise weniger komplexen Dienstes über die gleiche Funktionalität gelöst werden. Jeweils der aktuelle Zustand wird vom Modul an die PFE gemeldet. Die grundlegende Kommunikation zur dienstbasierten Steuerung in modularen verfahrenstechnischen Anlagen kann Abbildung 1 entnommen werden.

2.1.1 Zustandsautomat

2.1.1.1 Struktur des Automaten

Der Zustandsautomat eines jeden Dienstes besteht aus 16 Zuständen in vier Ebenen. Diese Zustände werden durch einen Befehl (Anforderung eines Zustandsübergangs) erreicht und wechseln selbstständig nach Beendigung der in ihnen hinterlegten Funktion in den Folgezustand. Von diesem Ablauf wird abgewichen sollte ein Befehl einer höheren Hierarchieebene des Automaten erfolgen. Der Zustandsautomat ist in vier Hierarchieebenen eingeteilt. In Ebene 1 liegen die Zustände *Starting*, *Running*, *Completing*, *Pausing*, *Paused* und *Resuming*. In Ebene 2 liegen die Zustände *Idle*, *Completed*, *Resetting*, *Holding*, *Held* und *Unholding* sowie Ebene 1. In Ebene 3 liegen die Zustände *Stopping* und *Stopped* sowie die Ebene 2. In Ebene 4 liegen die Zustände *Aborting* und *Aborted* sowie Ebene 3. Die Hierarchieebenen beschreiben, dass aus einer niedrigeren Ebene zu jedem Zeitpunkt der Wechsel in eine höhere Ebene erfolgen kann. Der Wechsel in Ebene 2 erfolgt hierbei über den Befehl *Hold*, der Übergang in Ebene 3 erfolgt über den Befehl *Stop*, der Übergang in Ebene 4 erfolgt über den Befehl *Abort*. Die Struktur des Zustandsautomaten ist in Abbildung 2 zu sehen. Hierbei sind die Befehle an den Pfeilen angegeben. Befehle in Rot sind nicht von der PFE anforderbar, sondern werden modulintern aufgelöst. Der Zustandsübergang SC bedeutet „State Change“ und zeigt an, dass dieser Übergang durch den vorgelagerten Zustand ausgelöst wird. Diese Art des Zustandsüberganges ist ausschließlich nach transienten Zuständen vorgesehen. Der Zustandsautomat besteht aus transienten und nicht transienten Zuständen. Hierbei handelt es sich bei allen auf „-ing“ endenden Zuständen mit der Ausnahme von *Running* um transiente Zustände.

Die durch die PFE anforderbaren Zustandsübergänge (Befehle) sind: *Start*, *Complete* (nur bei kontinuierlichen Fahrweisen, siehe Abschnitt 2.1.4.2), *Reset*, *Pause*, *Resume*, *Unhold*, *Stop*, *Abort* und

Restart. Abbildung 2 stellt den Automaten graphisch dar. Das Modul darf intern jederzeit Zustandsübergänge selbst auslösen. Der PFE wird lediglich der neue Zustand signalisiert.

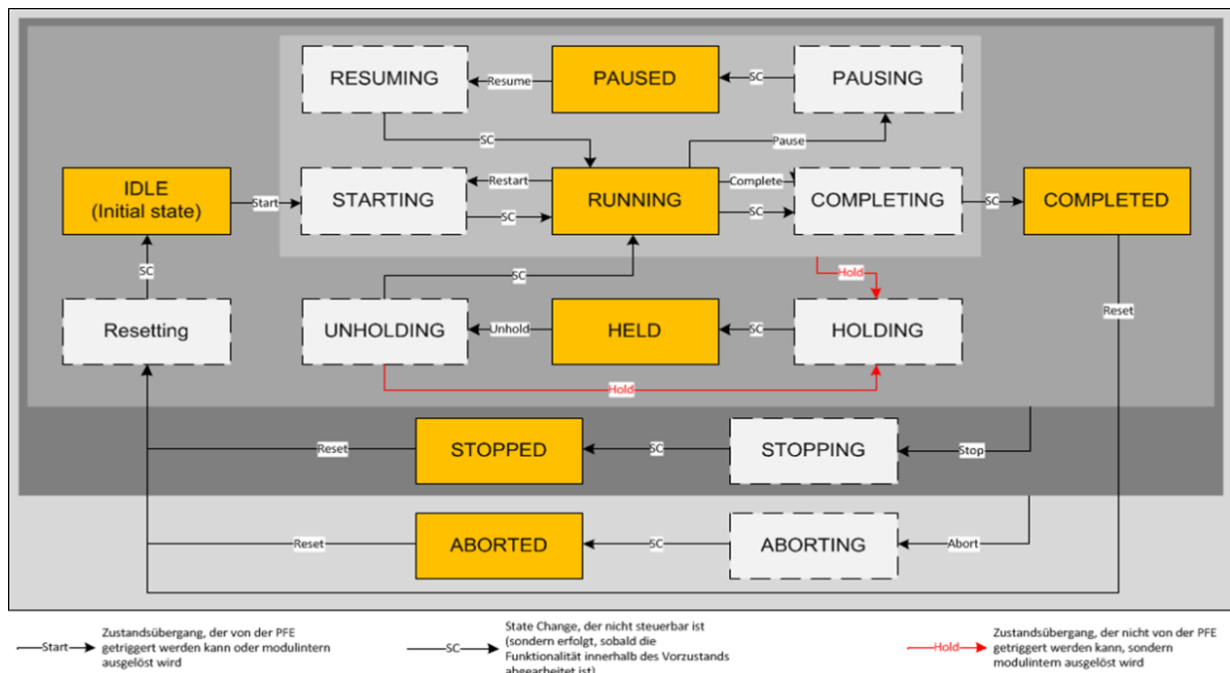


Abbildung 2: Dienst- Zustandsautomat

Das Zustandsmodell ist nicht konfigurierbar, d.h. jeder Dienst muss dieses vollständig implementieren (Zustände und Zustandsübergänge). Zustände können hierbei leer sein, müssen allerdings kurzzeitig beim Durchlaufen und einmalig angezeigt werden. In mehreren Zuständen darf die gleiche Funktionalität implementiert sein, z.B. in Stopping und Aborting.

2.1.1.2 Semantik der Zustände und Übergänge

Um die Dienste entsprechen ansteuern zu können, ist die Semantik der einzelnen Zustände festgelegt:

- *Idle*: Ein Dienst ist im Zustand *Idle*, wenn er bereit ist. *Idle* ist der Initialzustand des Dienstes.
- *Running*: Ein Dienst im Zustand *Running* führt die prozesstechnische Funktion(en) des Dienstes aus. Dies bedeutet, dass im Zustand *Running* die Hauptfunktionalität aus verfahrenstechnischer Sicht implementiert ist, z.B. führt ein Dienst „Mischen“ in *Running* die Funktion Mischen aus, die aus dem Vermischen mehrere Edukte besteht. Hierfür werden die entsprechenden Aktoren wie Ventile und Antriebe geschaltet.
- *Starting*: Der Zustand *Starting* beschreibt, dass Anfahren des Dienstes hin zu *Running*. In *Starting* werden die notwendigen Aktoren angefahren und alle Aktoren in die Grundstellung für *Running* versetzt.
- *Completing*: Der Zustand *Completing* beschreibt das geplante Abfahren eines Dienstes. Ist die Funktionalität in *Running* abgeschlossen oder wurde bei kontinuierlichen Fahrweisen durch den Bediener oder die Orchestrierung das Beenden der Funktionalität angefordert, so wird eine Abfahrunktionalität ausgeführt.
- *Completed*: Im Zustand *Completed* hat der Dienst alle Funktionalitäten beendet und es werden keine weiteren Funktionalitäten mehr ausgeführt.
- *Resetting*: Durch den Befehls *Reset* wird der Zustand *Resetting* erreicht, der alle Aktoren wieder in die Grundstellung für den Bereit-Zustand *Idle* verfährt. In *Resetting* können auch notwendige Zwischenreinigungen oder Ähnliches abgefahren werden. Der Befehl *Reset* darf nicht vom Modul selbst, sondern ausschließlich vom Bediener oder der PFE angefordert werden.

Die nachfolgenden Zustände können in Schleifen zusammengefasst werden:

- *Pause-Schleife*: Die Pause-Schleife bestehend aus den Zuständen *Pausing*, *Paused* und *Resuming* dient zur Implementierung der Logik die ausgeführt wird, wenn der Bediener oder die PFE ein Pausieren des Dienstes anfordert. Die Pause-Schleife wird durch den Befehl *Pause*

angefordert und muss auch durch den Befehl *Resume* wieder verlassen werden, sofern nicht ein Befehl einer anderen Hierarchieebene ausgelöst wird.

- Hold-Schleife: Die Hold-Schleife bestehend aus den Zuständen *Holding*, *Held* und *Unholding* dient zur Implementierung der in einem Fehlerfall auszuführenden Logik. Hierbei führen nur Fehlerfälle, die für den Dienst relevant sind und die einen manuellen Eingriff durch den Bediener benötigen, zum Eintritt in die Hold-Schleife. Fehlerfälle, die durch die Modullogik selbst gelöst werden können, sind in den jeweils aktiven Zuständen zu lösen, in denen die Fehlerfälle auftreten. Der Befehl *Hold* kann nur durch das Modul und zu keiner Zeit durch die PFE oder den Bediener angefordert werden. Bei der Auslegung der Dienst ist darauf zu achten, dass aus vielen Zuständen in die Hold-Schleife gewechselt werden kann, die Schleife jedoch immer in *Running* zurückführt. In Fällen, in denen nach einer Rückkehr in *Running* aus der Hold-Schleife nicht die komplette Dienst-Funktionalität ausgeführt werden soll, ist im Zustand *Running* zunächst zu prüfen, ob ein Zustandsübergang nach *Pausing* oder *Completing* durch das Modul ausgelöst werden muss.
- Stop-Schleife: Die Stop-Schleife bestehend aus den Zuständen *Stopping* und *Stopped* dient zur Implementierung des Stoppens der Dienstfunktionalität bevor die Funktionalität abgeschlossen ist. Das Stoppen des Dienstes kann zu einer Beschädigung des Produktes jedoch nicht des Moduls führen. Die Stop-Schleife kann durch die PFE, den Bediener oder auch durch das Modul ausgelöst werden.
- Abort-Schleife: Die Abort-Schleife bestehend aus den Zuständen *Aborting* und *Aborted* dient zur Implementierung eines schnellen Abschaltens des Dienstes. Hierbei kann es zu Beschädigungen des Produktes und des Moduls kommen. Die Abort-Schleife kann durch die PFE, den Bediener oder auch durch das Modul ausgelöst bzw. angefordert werden.

2.1.1.3 Steuer- und Statuswörter

Um einen Dienst aus der PFE zu orchestrieren verwendet jeder Dienst ein Steuer- und ein Statuswort. Sowohl das Steuer- als auch das Statuswort folgen einer definierten Enumeration. Die Enumerationen sind in Abschnitt 2.2.2.6 dargestellt. Diese Enumeration sind zwingend zu verwenden. Hierbei werden die Zustände vom Dienst an die PFE und die Befehle von der PFE an den Dienst übermittelt.

2.1.2 Dienst-Bediener Interaktion

Die Ausführung eines Dienstes kann eine Interaktion mit dem Bediener erfordern. Hierfür werden Variablen innerhalb der Dienstschnittstelle festgelegt. Die Modellierung erfolgt über ein Frage-Antwortprinzip und wird in 2.2.2.6 näher erläutert.

Zusätzlich kann die manuelle Bedienung einer modularen Anlage eine Interaktion mit dem Bediener erfordern. Um dies zu ermöglichen werden Konzepte zur Übersetzung von ganzzahligen Parametern in menschenlesbare Texte, Abschnitt 2.1.6.1, verwendet und der Dienst mit einem Attribute zur Anzeige des aktuellen Status der Ausführung ausgestattet, Abschnitt 2.2.2.4.

2.1.3 Zeitweise Verriegelung von Zustandsübergängen

Das Modul darf aufgrund vorliegender Prozesswerte oder Verriegelungen auf Einzelsteuerebene Zustandsübergänge zeitweise verriegeln. Hierfür wird für jeden Zustandsübergang ein Bit in der Variable *ControlEnable* angegeben. Dieses Bit zeigt an ob ein Zustandsübergang aktuell möglich ist. Ein gesetztes Bit entspricht einer Freigabe eines Befehls. Ist das Bit nicht gesetzt führt dies zur Verriegelung eines Befehls. Die Codierung der *ControlEnable*-Variable erfolgt bitweise. Die Codierung kann Abschnitt 2.2.2.7 entnommen werden.

Gibt der Wert von *ControlEnable* an, dass ein Befehl nicht anforderbar ist, so kann dieser Befehl nicht durch den Bediener oder die PFE ausgelöst werden, es muss zunächst der Befehl wieder freigegeben werden. Nur das Modul darf die Variable *ControlEnable* schreiben, die PFE und der Bediener haben ausschließlich Leserechte.

2.1.4 Fahrweisen

Um die Dienste der Module möglichst effizient schneiden zu können, können Fahrweisen genutzt werden. Dies ermöglicht zum einen eine Zuordnung von Prozesswerten die zum Betrieb des Dienstes notwendig sind und zum anderen die Fahrweise des Dienstes selbst. Fahrweisen können zu jeder Zeit

verändert werden, der Dienst übernimmt die aktuelle Fahrweise allerdings immer nur im Zustand *Starting*.

Des Weiteren erfolgt die Unterscheidung von Diensten in selbstbeendende und kontinuierliche Dienste. Diese Unterscheidung erfolgt auf Ebene der Fahrweisen. Hat ein Dienst nur eine Fahrweise, die selbstbeendend ist, ist der gesamte Dienst selbstbeendend. Ist diese Fahrweise kontinuierlich, so ist der Dienst kontinuierlich. Unterschiedliche Fahrweisen innerhalb eines Moduls sind erlaubt. Die PFE muss Dienste mit verschiedenen Fahrweisen unterstützen. Innerhalb eines Moduls können Dienste so geschnitten sein, dass keine unterschiedlichen Fahrweisen für einen Dienst vorliegen.

Die Information, ob eine Fahrweise selbstbeendend oder kontinuierlich arbeitet wird durch eine Variable, wie in 2.2.3 dargestellt, abgebildet. Jede Fahrweise kann zu Diagnosezwecken ihren aktuellen Gesundheitszustand anzeigen. Hierfür wird die Modellierung aus 2.2.3 verwendet.

2.1.4.1 Selbstbeendende Fahrweisen

Bei einer selbstbeendenden Fahrweise wird der Schaltbefehl zum wunschgemäßen Beenden des Dienstes (Zustandsübergang *Running* -> *Completing*; siehe 2.1.1) modulintern ermittelt. Der Zustandsübergang wird innerhalb des Moduls geschaltet. Die PFE oder der Bediener können den Zustandsübergang über die Zustände *Completing* und *Completed* nachvollziehen. Eine Beendigung durch externen Zugriff ist nur mit Hilfe der Zustandsübergänge *Stop* und *Abort* gegeben

2.1.4.2 Kontinuierliche Fahrweisen

Bei kontinuierlichen Fahrweisen wird der Befehl zum Beenden des Dienstes – in Abhängigkeit der Dienstbetriebsart nach 2.1.5 – von der PFE oder dem Bediener an das Modul gesendet.

2.1.5 Dienstbetriebsarten

Die Betriebsart von Diensten wird wie auf der Einzelsteuerebene, vgl. [VDI/VDE/NAMUR 2658 Blatt 3], über *OperationMode* gesteuert. Hierzu wird die Schnittstelle *ExtendedOperationMode*, wie in [VDI/VDE/NAMUR 2658 Blatt 3] definiert verwendet. Der Unterschied besteht darin, dass aus der Betriebsart *Automatic* auch direkt in die Betriebsart *Offline*, und umgekehrt, gewechselt werden darf (siehe Abbildung 3). Dies ist auf Einzelsteuerebene nicht möglich.

Jeder Dienst hat einen Zustandsautomat, wie in 2.1.1 beschrieben, in welchem je nach Betriebsart entweder vom Bediener, der übergeordneten Ablaufsteuerung oder einem anderen Dienst im Modul Zustandsübergänge angefordert werden dürfen.

Zur Umschaltung zwischen den möglichen Quelle (PFE, modulintern oder Bediener) für einen Dienst wird das gleiche Konzept genutzt wie bei den CDTs aus [VDI/VDE/NAMUR 2658 Blatt 3]:

- *OpMode* zur Umschaltung zwischen *Offline* - *Manual* - *Automatic_External* (PFE) - *Automatic_Internal* (modulintern).
- Die Umschaltung zwischen *Automatic_Internal* und *Automatic_External* wird über die Schnittstelle *SourceMode*, wie bereits in [VDI/VDE/NAMUR 2658 Blatt 3] definiert, realisiert.

In der Betriebsart *Offline* ist der Dienst als Gesamtes nicht betriebsbereit. Die unterlagerten, vom Dienst genutzten Aktoren können in *Manual* überführt werden, so dass diese für eine mögliche Inbetriebnahme durch den Bediener bedient werden können. Beim Verlassen der Betriebsart *Offline* überführt der Dienst die unterlagerten, vom Dienst genutzten Aktoren in die Betriebsart *Automatic* (Anforderung der Umschaltung ohne Rückmeldung). Diese können somit durch den Dienst verwendet werden.

In der Betriebsart *Manual* wird der Dienst selbst vom Bediener über die PFE (*OSLevel* > 0) oder ein lokales Panel/Display (*OSLevel* = 0) bedient. In der Betriebsart *Automatic_External* werden die Zustandsübergänge durch die PFE bedient. In der Betriebsart *Automatic_Internal* werden die Zustandsübergänge modulintern ausgelöst.

Das Modul / der Dienst kann stets eine Umschaltung der Betriebsart des Dienstes veranlassen.

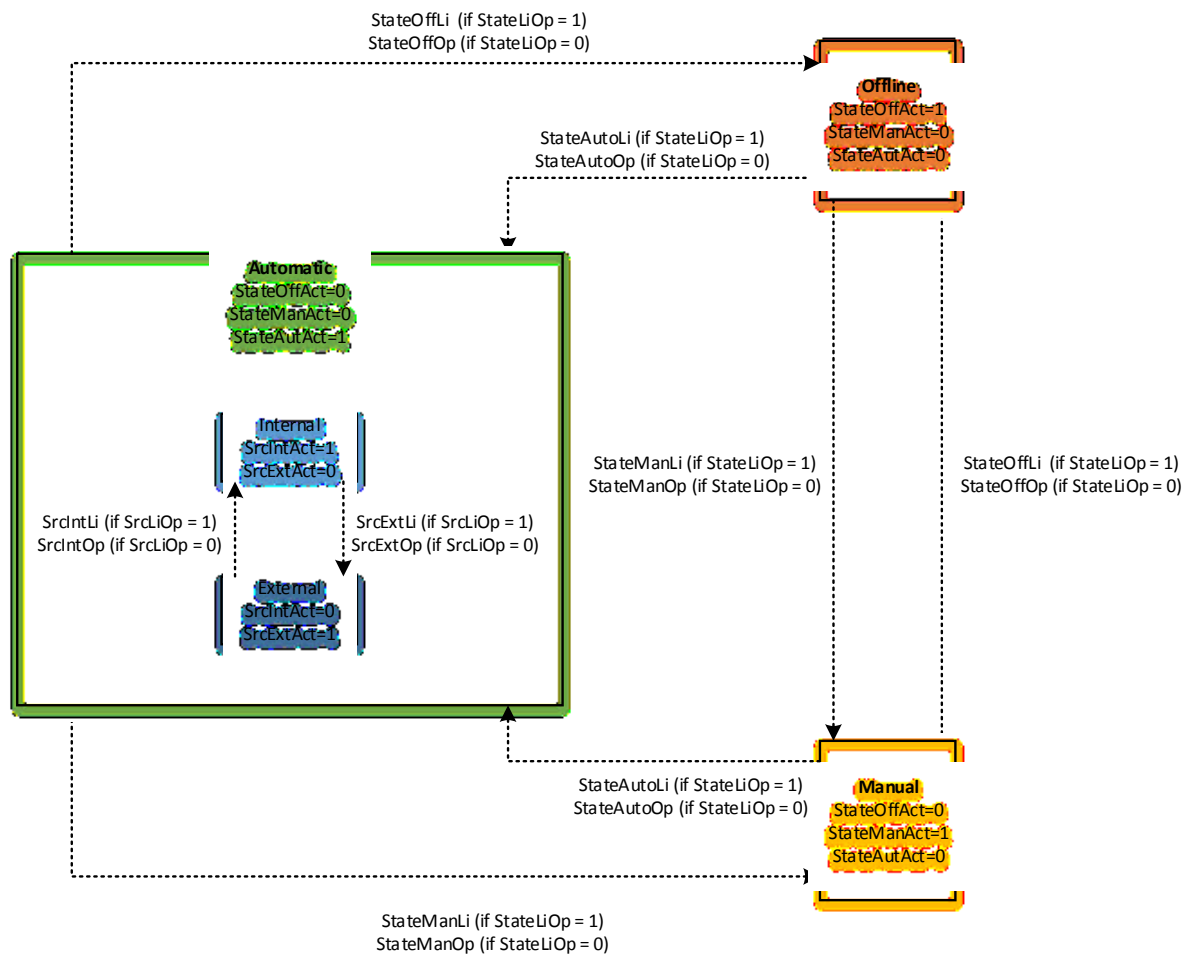


Abbildung 3: Betriebsarten eines Dienstes

2.1.6 Dienstparameter

Dienste können durch Parameter näher spezifiziert werden. Die Parameter können z.B. Zeiten oder Prozessparameter, wie Temperaturen und Drücke sein. Des Weiteren können Parameter notwendig sein, die den Dienst in der Ausführung näher spezifizieren. Sollwerte werden einem Dienst zugeordnet. Einzelne Sollwerte können in bestimmten Fahrweisen gültig sein, in anderen Fahrweisen nicht. Im MTP ist die Zuordnung, welche Fahrweise welche Sollwerte nutzt, abgebildet. Parameter werden in verschiedene Arten unterschieden, welche im Folgenden näher beschrieben werden.

Zur Parametrierung von Diensten stehen verschiedene Parametertypen zur Verfügung:

- Konfigurationsparameter, siehe 2.1.6.1
- Fahrweisenparameter, siehe 2.1.6.3
- Prozesswerte, siehe 2.1.6.4
- Reportwerte, siehe 2.1.6.5

Parameter werden immer einem Dienst zugeordnet. Dienstparameter können auf folgende Datentypen zum Wertaustausch verweisen:

- Binary (Binärwert)
- Digital (Ganzzahl)
- Analog (Gleitkommazahl)

Darüber hinaus existiert für den Reportwert noch ein Textformat:

- String (Text)

2.1.6.1 Verwendung Digitaler Parameter

Unabhängig des Parametertyps, können digitale Parameter als Enumerationen verwendet werden. Hierfür werden wie für die Dienst-Bediener Interaktion Listen angelegt. Der Index eines Texts innerhalb einer Liste wird zur Laufzeit in den digitalen Parameter geschrieben. Die Texte können dem Bediener angezeigt werden oder im Engineering verwendet werden um die Enumeration menschenlesbar zu gestalten. Die Modellierung der Listen wird in Abschnitt 2.2.5 beschrieben.

2.1.6.2 Konfigurationsparameter

Konfigurationsparameter werden verwendet um grundlegende Einstellungen für einen Dienst vorzunehmen, bevor dieser ausgeführt wird. Die Konfigurationsparameter gehören immer zu Diensten, können in mehreren Diensten verwendet werden und werden immer dann vom Dienst übernommen, wenn dieser zur Ausführung vorbereitet wird, aus der Betriebsart *Offline* in eine andere wechselt.

Während ein Dienst ausgeführt wird, bzw. in einer anderen Betriebsart als *Offline* ist, dürfen diese Parameter nicht verändert werden, bzw. eine Änderung wird erst übernommen wenn der Dienst wieder in der Betriebsart *Offline* war und diese sich wieder geändert hat. Konfigurationsparameter können nicht einzelnen Fahrweisen zugeordnet werden. Parameteränderungen können beim Wechsel der Betriebsart zur Dokumentation mitgeschnitten werden.

Beispiele für Konfigurationsparameter sind: Auswahl zu nutzender Sensoren als Eingangswerte, Grenzwerte vor- und nachgelagerter Module, Parameter zur Berechnung von Inertisierungszyklen.

2.1.6.3 Fahrweisenparameter

Fahrweisenparameter sind einzelnen Fahrweisen zugeordnet. Diese werden eingestellt bevor ein Dienst vom Zustand *Idle* gestartet oder aus dem Zustand *Running* neu gestartet wird. Immer wenn der Dienst im Zustand *Starting* ist, werden die Parameter übernommen und sind ab diesem Zeitpunkt gültig.

Parameter dieses Typs sind Rezeptrelevant und können sich während der Dienst ausgeführt wird ändern. Parameter dieses Typs können einer oder mehreren Fahrweisen zugeordnet werden. Änderungen der Fahrweisenparameter können zur Dokumentation immer im Zustand *Starting* mitgeschnitten werden.

Beispiele für Fahrweisenparameter sind: Rezeptrelevante Sollwerte (Temperaturvorgaben, Durchflussvorgaben, Druckvorgaben ...), Reglerparameter (Verstärkung, Nachhaltezeit, ...).

2.1.6.4 Prozesswerte

Prozesswerte sind Parameter die kontinuierlich gelesen oder geschrieben werden, also unabhängig der Betriebsart oder dem aktuellen Zustand eines Dienstes Werte bereitstellen oder erwarten. Prozessvariablen können einem oder mehreren Diensten zugeordnet werden, jedoch nicht den Fahrweisen.

Die Begriffe lesend und schreibend bezieht sich auf die Perspektive aus Sicht der PFE. Ein lesender Prozesswert wird von einem Modul für externen Zugriff bereitgestellt und kann von der PFE an ein anderes Modul weitergeleitet werden. Ein zu schreibender Prozesswert wird von der PFE in ein Modul geschrieben.

Für die horizontale Module-zu-Modul Kommunikation, werden Parameter dieses Typs verwendet. Hierbei kann jedem zu schreibenden Prozesswert ein zu lesender Prozesswert übertragen werden. Diese Übertragung realisiert heute die PFE.

Beispiele für Prozessvariablen sind: Modulübergreifenden Verriegelungen, modulübergreifenden Regelungen (Sollwert aus Modul A, Regelung und Stellaktor in Modul B).

2.1.6.5 Reportwerte

Ein Reportwert ist ein Wert der zu Nachweis- und Dokumentationspflichten eines Anlagenbetreibers von einem Dienst zur Verfügung gestellt wird. Ein Reportwert wird kontinuierlich aktualisiert. In den Zuständen *Completed*, *Aborted* und *Stopped*, wird der Wert eingefroren, damit dieser von einem angeordneten Dokumentations- und oder Archiv-System gelesen werden kann.

2.1.7 Dienstabhängigkeiten

Dienstabhängigkeiten können zwischen Diensten oder zwischen Dienst und Einzelsteuerebene beschrieben werden. Hierfür wird in Abschnitt 2.2.6 die Modellierung gezeigt. Hinzu kommen Verriegelungen auf der Einzelsteuerebene.

2.1.7.1 Verwendetes Anlagenequipment

Jedem Dienst wird eine Liste des darin verwendeten Anlagenequipments zugeordnet. Diese beinhaltet das verwendete Equipment, sowie den im Dienst nötigen Zustand (z.B. offen oder geschlossen). Die Modellierung wird in 2.2.6 dargestellt.

2.1.7.2 Enable

Eine *Enable* Abhängigkeit besitzt stets einen Zustand eines Dienstes als Quelle und einen Zustandsübergang eines anderen Dienstes des gleichen Moduls als Ziel. Die Funktion ist so zu verstehen, dass der Ziel-Zustandsübergang ausschließlich dann freigegeben ist, wenn der Quell-Zustand aktiv ist. Die Freigabe des Zustandsübergangs ist über *ControlEnable* zu modellieren. Diese Dienste-Abhängigkeit dient der Freigabe eines Zustandsüberganges durch eine zwingend vorliegende Vorbedingung repräsentiert durch einen Zustand eines Dienstes. [J. Ladiges, A. Köcher, P. Clement, H. Bloch, T. Holm, P. Altmann, A. Fay, L. Urbas: Entwurf, Modellierung und Verifikation von Serviceabhängigkeiten in Prozessmodulen. In: at – Automatisierungstechnik, 66(5), 418-437. DOI: <https://doi.org/10.1515/auto-2017-0076>]

2.1.7.3 Disable

Eine *Disable* Abhängigkeiten besitzt stets einen Zustand eines Dienstes als Quelle und einen Zustandsübergang eines anderen Dienstes des gleichen Moduls als Ziel. Die Funktion ist so zu verstehen, dass der Ziel-Zustandsübergang nicht freigegeben (verriegelt) ist, wenn der Quell-Zustand aktiv ist. Die Freigabe des Zustandsübergangs ist über *ControlEnable* zu modellieren. Diese Dienste-Abhängigkeit dient der Verriegelung eines Zustandsübergangs eines Dienstes durch einen Zustand eines anderen Dienstes. [J. Ladiges, A. Köcher, P. Clement, H. Bloch, T. Holm, P. Altmann, A. Fay, L. Urbas: Entwurf, Modellierung und Verifikation von Serviceabhängigkeiten in Prozessmodulen. In: at – Automatisierungstechnik, 66(5), 418-437. DOI: <https://doi.org/10.1515/auto-2017-0076>]

2.1.7.4 Synchronisation

Eine *Sync*-Abhängigkeit besitzt stets einen Zustandsübergang eines Dienstes als Quelle und einen Zustandsübergang eines anderen Dienstes des gleichen Moduls als Ziel. Der Ziel-Zustandsübergang wird hierdurch ebenfalls durchgeführt, sobald der Quell-Zustandsübergang durchgeführt wird. Diese Dienste-Abhängigkeit dient der Synchronisation von Zustandsübergängen verschiedener Dienste. [J. Ladiges, A. Köcher, P. Clement, H. Bloch, T. Holm, P. Altmann, A. Fay, L. Urbas: Entwurf, Modellierung und Verifikation von Serviceabhängigkeiten in Prozessmodulen. In: at – Automatisierungstechnik, 66(5), 418-437. DOI: <https://doi.org/10.1515/auto-2017-0076>]

2.1.7.5 Force

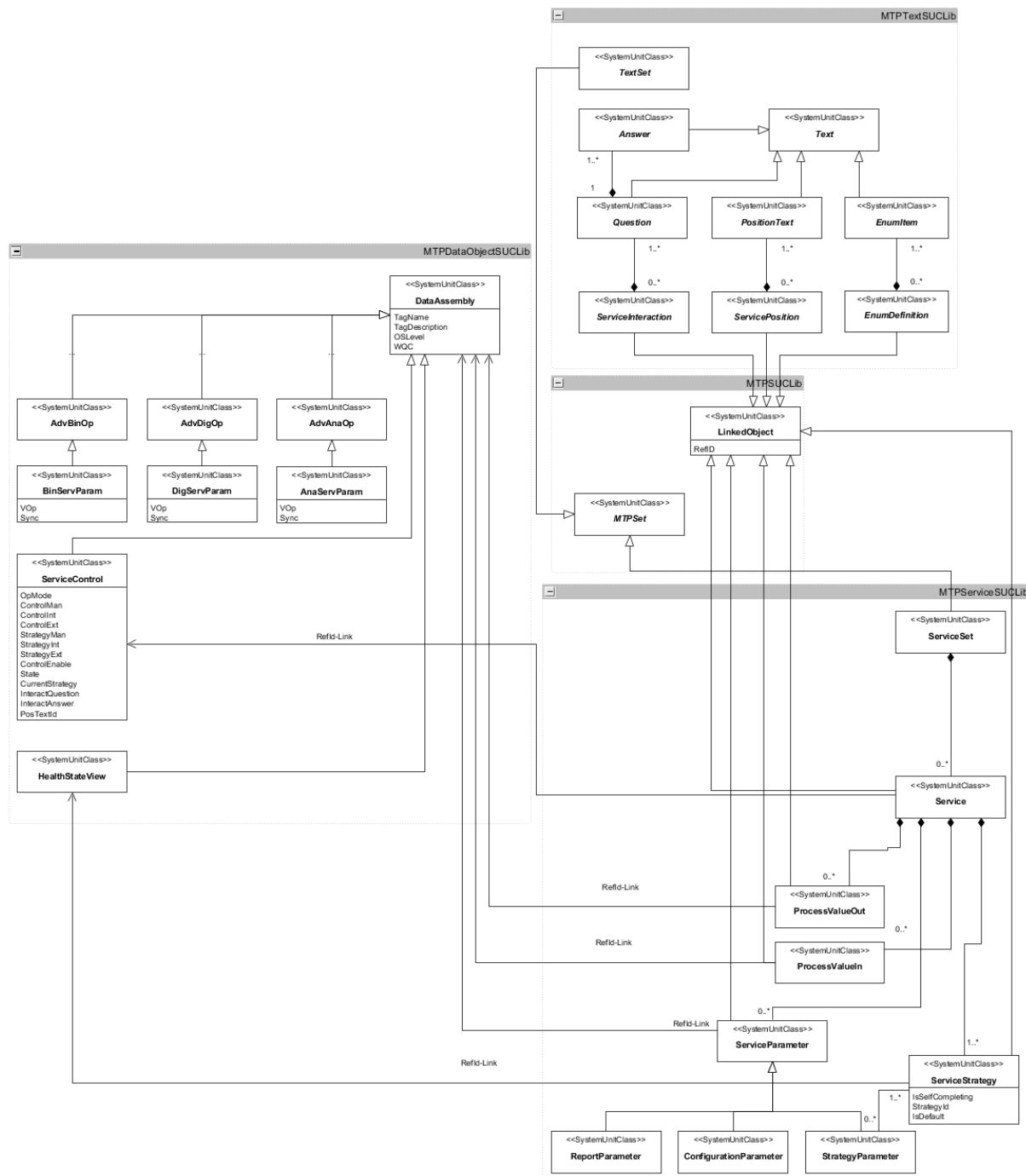
Eine *Force*-Abhängigkeit besitzt stets einen Zustand eines Dienstes als Quelle und einen Zustandsübergang eines anderen Dienstes des gleichen Moduls als Ziel. Der Zielzustandsübergang wird durch die im Quell-Zustand hinterlegte Logik angesteuert. Diese Dienste-Abhängigkeit dient dem Dienstaufwurf aus einem expliziten Zustand eines Dienstes heraus. Die Force-Abhängigkeit unterscheidet sich gegenüber der *Sync*-Abhängigkeit im Zeitpunkt der Ansteuerung des Ziel-Zustandsüberganges. [J. Ladiges, A. Köcher, P. Clement, H. Bloch, T. Holm, P. Altmann, A. Fay, L. Urbas: Entwurf, Modellierung und Verifikation von Serviceabhängigkeiten in Prozessmodulen. In: at – Automatisierungstechnik, 66(5), 418-437. DOI: <https://doi.org/10.1515/auto-2017-0076>]

2.2 Dienstmodellierung

2.2.1 Übersicht Dienstmodell

Die Darstellung des Aspekts der dienstbasierten Prozessführung erfolgt als Erweiterung der Richtlinienserie, wie in [VDI/VDE/NAMUR 2658 Blatt 1] beschrieben. Hierfür wird die Bibliothek *MTPServiceSUCLib* verwendet. Das zugehörige UML-Modell der notwendigen *SystemUnitClasses* ist in Abbildung 4 dargestellt, wobei aus Gründen der Übersichtlichkeit Bibliotheken und

Abbildung 4: SystemUnitClasses für die Beschreibung der Dienstschnittstelle



Ein *ServiceSet* kann beliebig viele Dienste (*Services*) besitzen, wobei jeder dieser Dienste mindestens eine *ServiceStrategy* beinhalten muss. Des Weiteren können einem Dienst (*Service*) *ServiceParameter* zugeordnet sein. Hierbei muss eine Unterscheidung zwischen den Ableitungen *ConfigurationParameter*, *ReportParameter* und *StrategyParameter* getroffen werden. Die *SystemUnitClasses* *Service* und *ServiceParameter* sind von dem in [VDI/VDE/NAMUR 2658 Blatt 1] definierten *LinkedObject* abgeleitet. Durch diese Ableitung werden die *ServiceParameter* mit den in [VDI/VDE/NAMUR 2658 Blatt 3] definierten Datenstrukturen detailliert beschrieben, da über einen ID-Link ein Verweis auf ein zugehöriges *DataAssembly* hergestellt wird.

Weiterhin wird über einen ID-Link die Referenz zwischen dem Dienst (*Service*) und dem korrespondierenden *ServiceControl InternalElement* beschrieben. Die *ServiceControl SystemUnitClass* erbt *DataAssembly* aus [VDI/VDE/NAMUR 2658 Blatt 3], wodurch auch die Attribute *TagName*, *TagDescription*, *OSLevel* und *WQC* vererbt werden. Die Erweiterung um die zusätzlichen Attribute *OpMode*, *ControlMan*, *ControlInt*, *ControlExt*, *StrategyMan*, *StrategyInt*, *StrategyExt*, *ControlEnable*, *State*, *InteractQuestion*, *InteractAnswer*, *PosTextId* und *CurrentStrategy* sowie deren Bedeutung wird in Abschnitt 2.2.2 erläutert.

Jede *ServiceStrategy* kann mehrere *StrategyParameter* besitzen (hierarchisch unterhalb des *InternalElement* von *ServiceStrategy*). Die weiteren Attribute kennzeichnen, ob es sich um eine selbstbeendende Strategie (*IsSelfCompleting*) handelt, wie die ID der Strategie (*StrategyId*) lautet und ob es sich bei dieser Strategie um die standardmäßig auszuführende Strategie (*IsDefault*) handelt.

Prozesswert (lesend oder schreiben) werden als *ProcessValueIn* und *ProcessValueOut InternalElements* hierarchisch unterhalb des Dienstes (*Service*) angehängt. Genau wie die Konfigurations-, Fahrweisen- und Reportparameter werden diese über einen ID-Link mit der *InstanceList* des Manifests verbunden.

2.2.2 Modellierung der Dienstschnittstelle

Zur Kommunikation besitzt ein Dienst eine Schnittstelle bestehend aus mehreren Parametern. Diese werden verwendet um den Dienst zu Steuern und Rückmeldung über den aktuellen internen Zustand des Dienstes zu erhalten.

Als Basisklasse wird *DataAssembly*, wie in [VDI/VDE/NAMUR 2658 Blatt 3] definiert, verwendet. Dies erlaubt eine Abbildung der Dienstschnittstelle im Kommunikationsaspekt des Manifests nach [VDI/VDE/NAMUR 2658 Blatt 1]. Zusätzlich werden vier Parameter geerbt, welche den Dienst beschreiben:

- *TagName* enthält den Namen des Dienstes
- *TagDescription* enthält eine Beschreibung des Dienstes
- *OSLevel* wird zur Freigabe einer bestimmten Bedienstation verwendet
- *WQC* beschreibt die Qualität der übermittelten Werte des Dienstes

Diese Parameter werden in [VDI/VDE/NAMUR 2658 Blatt 3] bereits beschrieben und werden für den Dienst analog zur Einzelsteuerebene verwendet.

Hinzu kommen spezifische Parameter zur Steuerung des Dienstes. Eine Übersicht der beinhalteten Parameter ist in Tabelle 1 gegeben.

Tabelle 1: Schnittstellendefinition eines Dienstes

Name: ServiceControl			
Parent Interface			
MTPDataObjectSUCLib/DataAssembly			
Attribute	Access	Type	Description
OpMode	PFE ↔ MOD	DWORD	See operation mode [VDI/VDE/NAMUR 2658 Blatt 3] - ExtendedOperationMode and Source Mode - Offline (StateManAct = false & StateAutAct = false) - Manual (StateManAct = true) - Automatic_Internal (StateAutAct = true & SrcIntAct = true) - Automatic_External (StateAutAct = true & SrcExtAct = true)
ControlMan	PFE → MOD	DWORD	Command variable – Manual Control - Encoding see definition of commands - Only relevant if StateManAct is active
StrategyMan	PFE → MOD	DWORD	Strategy variable – Manual - Only relevant if StateManAct is active
ControlInt	PFE ← MOD	DWORD	Command Variable – module internal - Encoding see definition of Commands - Only relevant if StateAutAct and SrcIntAct is active
StrategyInt	PFE ← MOD	DWORD	Strategy Variable – module internal - Only relevant if StateAutAct and SrcIntAct is active
ControlExt	PFE → MOD	DWORD	Command Variable - External Control System - Encoding see definition of commands - Only relevant if StateAutAct and SrcExtAct is active
StrategyExt	PFE → MOD	DWORD	Strategy Variable - External Control System - Only relevant if StateAutAct and SrcExtAct is active

Name: ServiceControl			
Parent Interface			
MTPDataObjectSUCLib/DataAssembly			
Attribute	Access	Type	Description
State	PFE ← MOD	DWORD	State Variable - Encoding see definition of State
ControlEnable	PFE ← MOD	DWORD	Control Enable Variable - Encoding see definition of ControlEnable
CurrentStrategy	PFE ← MOD	DWORD	Currently active strategy of the service: - set during transition from Idle to Starting - reset during transition from Resetting to Idle
PosTextId	PFE ← MOD	DWORD	Id of the text in the corresponding text list, as described in 2.2.5
InteractQuestion	PFE ← MOD	DWORD	Id of the question for service interaction in the corresponding text list, as described in 2.2.5
InteractAnswer	PFE → MOD	DWORD	Id of the answer for service interaction in the corresponding text list, as described in 2.2.5

2.2.2.1 Betriebsarten

Jeder Dienst besitzt einen Parameter zur Steuerung der Betriebsart. Dieser Parameter – *OpMode* – verwendet die Betriebsartenumschaltung wie in [VDI/VDE/NAMUR 2658 Blatt 3] und Abschnitt 2.1.5 beschrieben. Ein Dienst verwendet grundsätzlich die Betriebsartenumschaltung *ExtendedOperationMode* und *SourceMode*, wobei *ExtendedOperationMode* erweitert wird um einen direkten Übergang zwischen *Automatic* und *Offline*. Die Codierung der Betriebsartenumschaltung kann [VDI/VDE/NAMUR 2658 Blatt 3] entnommen werden. Die weiteren Parameter werden in den folgenden Abschnitten näher erläutert.

2.2.2.2 Steuerwörter

Zur Steuerung von Diensten werden drei Parameter bereitgestellt:

- *ControlMan*
- *ControlExt*
- *ControlInt*

Die drei Parameter bilden drei mögliche Kanäle zur Ansteuerung des Dienstes ab. So ist *ControlMan* zur Verwendung aus einem Bedienbild, bzw. durch den Bediener, vorgesehen, *ControlExt* zur Verwendung aus der PFE, z.B. aus einem Rezept, und *ControlInt* wird modulintern verwendet. Je nachdem, in welcher Betriebsart sich der Dienst befindet, wählt das Modul den zugehörigen Kanal aus. Die Betriebsarten können Tabelle 1 entnommen werden.

Die Parameter werden als Steuerwörter implementiert. Jedes Bit des Steuerworts steht hierbei für einen Befehl und es darf immer nur ein Bit des Steuerworts gesetzt sein. Die entsprechende Codierung kann Abschnitt 2.2.2.7 entnommen werden. Die Logik im Modul setzt den anliegende Befehl in den Steuerwörtern *ControlMan* und *ControlExt* zurück (Statusbits = 0), sobald er es ausgeführt hat. Somit wird verhindert, dass ein Befehl dauerhaft ansteht und ungewollt mehrfach ausgeführt wird. Dies kann z.B. dazu führen, dass der Dienst mehrmals gestartet wird, wenn die Befehle *Start*, *Complete* und *Reset* gesendet wurden und seitens der PFE anstehen bleiben.

2.2.2.3 Fahrweisen

Fahrweisen werden ebenfalls über die Dienstschnittstelle gesetzt. Jede Fahrweise hat eine modulweit eindeutige ID. Diese dient zur Identifizierung der Fahrweise und wird verwendet um während des Betriebs (bzw. beim Starten eines Dienstes) die richtige Fahrweise auszuwählen.

Um die Fahrweise durch verschiedenen Quellen (PFE, Modulintern, Bediener) auszuwählen, stehen drei Parameter zur Verfügung die je nach Betriebsart modulintern ausgewählt werden:

- *StrategyMan*
- *StrategyExt*
- *StrategyInt*

Die drei Parameter bilden drei mögliche Kanäle zur Ansteuerung des Dienstes ab. So kann *StrategyMan*, aus einem Bedienbild, bzw. durch den Bediener beschrieben werden, *StrategyExt* kann durch die PFE, z.B. aus einem Rezept beschrieben werden und *StrategyInt* wird modulintern verwendet.

Je nachdem, in welcher Betriebsart sich der Dienst befindet, wählt das Modul den zugehörigen Kanal aus. Die Betriebsarten können Tabelle 1 entnommen werden.

2.2.2.4 Rückmeldung

Zusätzlich zu den Steuerwörtern und den Parametern zum Setzen der Fahrweise meldet das Modul seinen aktuellen Status zurück. Dies geschieht über folgende Parameter:

- *CurrentStrategy*
- *State*
- *ControlEnable*
- *PosTextId*

CurrentStrategy beinhaltet die ID der derzeit ausgewählten Fahrweise. Diese ändert sich nur, wenn eine neue Fahrweise ausgewählt wird und der Dienst entsprechend neu gestartet wird.

Zusätzlich wird der aktuell anliegende Zustand des Dienstes angezeigt. Hierfür wird der Parameter *State* verwendet. *State* kann alle Zustände des Dienstes annehmen, muss jedoch immer im definierten Bereich liegen. Zustände außerhalb der Definition sind ungültig. Das Statuswort *State* wird wie die Steuerwörter als Bitfeld codiert, wobei immer genau ein Bit gesetzt sein muss. Die Codierung des Statusworts wird in Abschnitt 2.2.2.7 dargestellt.

Die Rückmeldung *ControlEnable* beinhaltet die verfügbaren Zustandswechsel des Dienstes, siehe auch 2.2.2.5.

Zusätzlich kann jeder Dienst über die Dienstschnittstelle der PFE Informationen über den aktuellen Stand der Bearbeitung informieren. Hierzu wird die Variable *PosTextId* verwendet. Diese Variable wird vom Dienst geschrieben. Im MTP sind die zugeordneten Texte in Listen organisiert. Die PFE muss eine Mindesttextlänge von 256 Zeichen unterstützen. Die PFE kann den Text in der Anzeige abschneiden. Die Modellierung von Textlisten ist in Abschnitt 2.2.5 beschrieben.

2.2.2.5 Anzeige mögliche Zustandswechsel während des Betriebs

Während des Betriebs eines Dienstes ist es möglich, dass definierte Zustandsübergänge zeitweise nicht angefordert werden dürfen (siehe Kapitel 2.1.3). Dies passiert unter Umständen wenn Elemente der Einzelsteuerebene einen Zustand halten, welcher dies verhindert und erst nach Auflösung dieses Zustands darf ein Zustandswechsel im Dienst angefordert werden.

Deshalb ist es notwendig während des Betriebs anzuzeigen, welche Übergänge in der aktuell anliegenden Betriebsart, im laufenden Dienst, bei aktueller Fahrweise und in Abhängigkeit zur Einzelsteuerebene im Moment möglich sind. Der Parameter *ControlEnable* wird hierfür verwendet.

Die Bits dieser Parameter werden analog zu den Bits der Steuerwörter codiert. Die Bits zeigen an, welcher Steuerbefehl im Moment an den Dienst gesendet werden darf. Hierbei bedeutet 1 das ein Steuerbefehl möglich ist, eine 0 das ein Setzen dieses Steuerbefehls unterbunden wird. In *ControlEnable* sind damit die Bits gesetzt, die aktuell möglich sind. Das Setzen mehrerer Bits ist damit möglich.

Die Codierung von *ControlEnable* befindet sich ebenfalls in Abschnitt 2.2.2.7.

2.2.2.6 Modul Bediener Interaktion

Die Interaktion erfolgt zur Laufzeit über die Variablen *InteractQuestion* und *InteractAnswer*. Jedem gültigen Wert dieser Variablen ist ein Text im MTP zugeordnet, der von der PFE angezeigt werden kann.

Zur Initiierung einer Interaktion setzt der Dienst die Variable *InteractQuestion* auf einen Wert ungleich 0. Der Wert der Variable zeigt auf den anzuzeigenden Text in der zugeordneten Liste. Diese Liste enthält den Fragetext und zusätzlich eine Liste möglicher Antworten für den Bediener.

Wählt der Bediener eine Antwort aus der Antwortliste aus, so wird der zugehörige Index der Liste in die Antwortvariable *InteractAnswer* geschrieben. Die Modellierung von Textlisten ist in Abschnitt 2.2.5 dargestellt.

2.2.2.7 Codierung der Steuer- und Statuswörter

Die Codierung der Variablen *State* und *Control* erfolgt bitweise. Die zugehörige Codierung, deren Übersetzung in einen ganzzahligen Wert und die Bedeutung wird in Tabelle 2 abgebildet. Beim Steuer- sowie beim Statuswort darf hierbei immer nur ein Bit gesetzt sein, während *ControlEnable* immer alle möglichen Zustandsübergänge anzeigt und damit mehrere Bits gesetzt sein dürfen.

Tabelle 2: Codierung der Steuer- und Status Wörter eines Dienstes

Codierung:								
States (State) 32Bit			Commands (Control) 32Bit			ControlEnable (32Bit)		
Bit No.	Int	Definition	Bit No.	Int	Definition	Bit No.	Definition	
0	1	Undefined	0	1	Undefined	0	Undefined	
1	2	Not Used	1	2	Reset	1	Reset	
2	4	Stopped	2	4	Start	2	Start	
3	8	Starting	3	8	Stop	3	Stop	
4	16	Idle	4	16	Not Used	4	Not used	
5	32	Paused	5	32	Unhold	5	Unhold	
6	64	Running	6	64	Pause	6	Pause	
7	128	Stopping	7	128	Resume	7	Resume	
8	256	Aborting	8	256	Abort	8	Abort	
9	512	Aborted	9	512	Restart	9	Restart	
10	1024	Holding	10	1024	Complete	10	Complete	
11	2048	Held	11		Reserve	11	Reserve	
12	4096	Unholding	12			12		
13	8192	Pausing	13			13		
14	16384	Resuming	14			14		
15	32768	Resetting	15			15		
16	65536	Completing	16 - 31			16 - 31		
17	131072	Completed						
18 - 31		Reserve						

2.2.3 Modellierung der Fahrweisen

Fahrweisen werden durch die in Tabelle 3 beschriebenen Parameter beschrieben. Jeder Dienst besitzt wenigstens eine Fahrweise.

Tabelle 3: Schnittstellendefinition einer Fahrweise

Name: ServiceStrategy			
Attribute	Access	Type	Description
IsSelfCompleting	PFE ← MOD	BOOL	True if the strategy is self completing.
StrategyId	PFE ← MOD	DWORD	The ID of the strategy e.g. 1
IsDefault	PFE ← MOD	BOOL	True if this strategy is default for the service.

Zusätzlich zu diesen Attributen verweist jede Fahrweise auf ein *InternalElement* der *SystemUnitClass HealthStateView* in der *InstanceList* des Kommunikationsaspekts. Dieses *InternalElement* zeigt über seinen *WQC* den aktuellen Gesundheitszustand der Fahrweise an. Dieser ergibt sich über die Gesundheitszustände der in der Fahrweise verwendeten Elemente der Einzelsteuerebene.

2.2.4 Modellierung der Dienstparametern

Je nach verwendeten Parametertyp, werden die Parameter unterschiedlich modelliert. Allen gemeinsam ist, dass in der *InstanceHierarchy* der Dienste immer eine Referenz auf ein *InternalElement* der *SystemUnitClass DataAssembly* innerhalb des Kommunikationsaspekts und der darin definierten Kommunikationsschnittstelle verwendet wird.

Die *SystemUnitClasses ConfigurationParameter*, *StrategyParameter* und *ReportParameter* werden von *ServiceParameter* abgeleitet und besitzen deshalb immer ein Attribut *RefId* welches wie in [VDI/VDE/NAMUR 2658 Blatt 1] beschrieben auf die zugehörige Kommunikationsschnittstelle verweist.

Als Sonderfall für Dienstparameter sind die Prozesswerte anzusehen. Diese werden direkt von *LinkedObject* abgeleitet, da diese nicht zwangsläufig nur innerhalb eines bestimmten Dienstes verwendet werden, somit streng genommen keine *ServiceParameter* sind.

2.2.4.1 Konfigurationsparameter

Konfigurationsparameter werden immer genau einem Dienst zugeordnet. Deshalb werden *InternalElements* dieses Typs von *ConfigurationParameter* gebildet und hierarchisch unterhalb des

Service InternalElement angehängt. Die *InternalElements* von *ConfigurationParameter* verlinken auf ein *InternalElement* von *DataAssembly* im Kommunikationsaspekt des Manifests.

Da Konfigurationsparameter von drei verschiedenen Typen sein dürfen und die gleichen Kommunikationskanäle wie die Dienstschnittstelle haben (Externer Zugriff durch die PFE, manueller Eingriff durch den Bediener, Interne Steuerung), verweisen die Konfigurationsparameter aus der *InstanceHierarchy* der Dienstes auf *AnaServParam*, *DigServParam* oder *BinServParam* welche von *DataAssembly* erben.

Die Betriebsart des CDTs bestimmt welcher Kommunikationskanal verwendet wird. Daher haben die CDTs die Betriebsarten *Manual* und *Automatic*, wobei in *Automatic* zwischen den Quellen *Internal* und *External* unterschieden wird. Die Umschaltung der Betriebsart erfolgt über die Variable *OpMode*.

Sobald ein Konfigurationsparameter geschrieben wird, prüft der Dienst, ob der Wert gültig ist. Bei einem gültigen Wert wird der *WQC* des Konfigurationsparameters auf *Good_Edited* gesetzt. Ist der Wert ungültig, wird der *WQC* auf *Bad* gesetzt. Solange ein Konfigurationsparameter ungültig ist, kann der Dienst nicht in die Betriebsarten *Manual* oder *Automatic* wechseln, d.h. der Dienst wird eine Umschaltung ablehnen. Nach dem Umschalten der Betriebsart des Dienstes wird der *WQC* aller Konfigurationsparameter auf *Good* gesetzt.

2.2.4.2 Fahrweisenparameter

Fahrweisenparameter können einer oder mehrerer Fahrweisen zugeordnet werden. Deshalb werden diese als *InternalElement* von *StrategyParameter* hierarchisch unterhalb eines *Strategy InternalElements* angehängt. Die *InternalElements* von *StrategyParameter* verlinken auf eine *InternalElement* von *DataAssembly* im Kommunikationsaspekt des Manifests.

Da Fahrweisenparameter von drei verschiedenen Typen sein dürfen und die gleichen Kommunikationskanäle wie die Dienstschnittstelle haben (Externer Zugriff durch die PFE, manueller Eingriff durch den Bediener, Interne Steuerung), verweisen die Fahrweisenparameter aus der *InstanceHierarchy* der Services auf *AnaServParam*, *DigServParam* oder *BinServParam* welche von *DataAssembly* erben.

Die Betriebsart des CDTs bestimmt welcher Kommunikationskanal verwendet wird. Daher haben die CDTs die Betriebsarten *Manual* und *Automatic*, wobei in *Automatic* zwischen den Quellen *Internal* und *External* unterschieden wird. Die Umschaltung der Betriebsart erfolgt über die Variable *OpMode*.

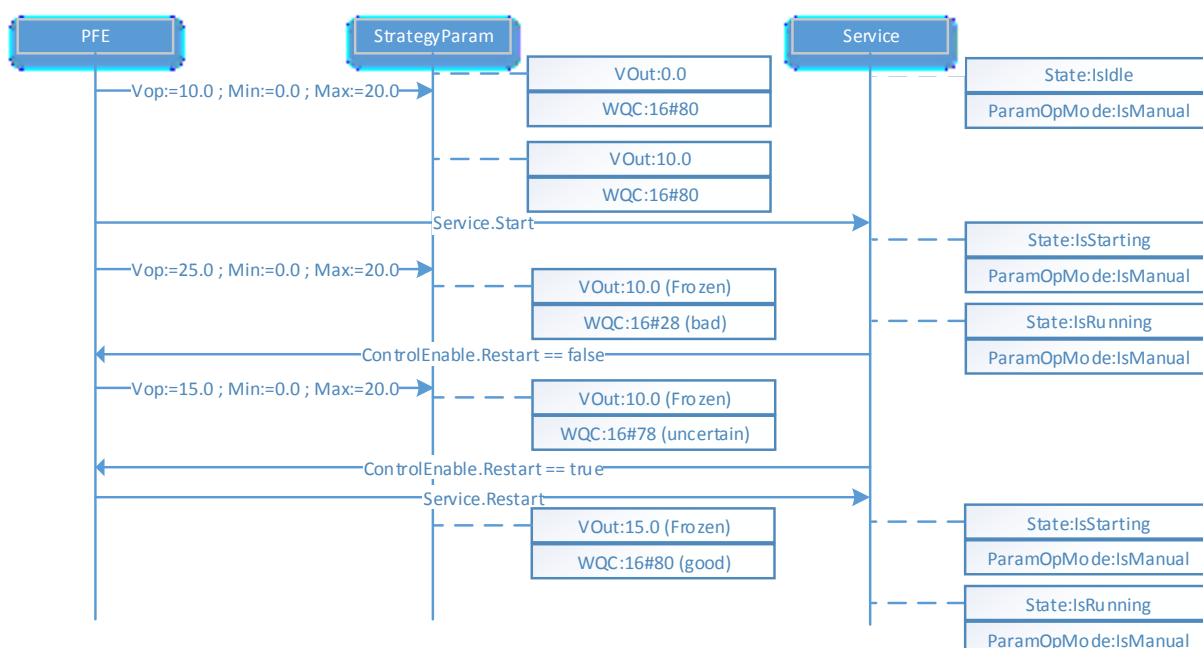


Abbildung 5: Übernahmesequenz von Fahrweisenparametern

Ein Fahrweisenparameter kann innerhalb mehrerer Fahrweisen eines Dienstes verwendet werden. Hierfür wird unterhalb jedes *Strategy InternalElements*, in welchen der Parameter verwendet werden

sollen, eine *StrategyParameter InternalElement* angehängt. *InternalElements* von *StrategyParameter* die auf die gleiche Kommunikationsschnittstelle verweisen, gehören zum gleichen Fahrweisenparameter.

Fahrweisenparameter werden vom Dienst beim Starten oder Neustarten überprüft und bei Zulässigkeit des Wertes übernommen. Sobald ein Fahrweisenparameter geschrieben wird, prüft der Dienst, ob der Wert gültig ist. Bei einem gültigen Wert wird der *WQC* des Fahrweisenparameters auf *Good_Edited* gesetzt. Ist der Wert ungültig, wird der *WQC* auf *Bad* gesetzt.

Solange ein Fahrweisenparameter ungültig ist, kann der Dienst nicht gestartet werden, d.h. der Dienst setzt die zugehörigen Bits von *ControlEnable* für *Start/Restart* auf *falsch*. Nach dem Befehl *Start/Restart* des Dienstes wird der *WQC* aller Fahrweisenparameter auf *Good* gesetzt. Dieser Ablauf ist in Abbildung 5 dargestellt.

2.2.4.3 Reportparameter

Reportparameter werden immer einem Dienst zugeordnet. Deshalb werden diese als *InternalElement* von *ReportParameter* hierarchisch unterhalb des *Service InternalElements* angehängt. Die *InternalElements* von *ReportParameter* verlinken auf ein *InternalElement* von *DataAssembly* im Kommunikationsaspekt des Manifests.

Auf *ReportParameter* kann PFE-seitig nur lesend zugegriffen werden, weshalb diese nicht auf die spezifischen Ableitungen, sondern auf *AnaView*, *DigView* oder *BinView* verlinken.

2.2.4.4 Prozesswerte

Prozesswerte sind innerhalb der Dienstparameter als Sonderfall zu betrachten. Prozesswerte können Dienstübergreifend verwendet werden und z.B. zur Modul-zu-Modul Kommunikation verwendet werden.

Zur Modellierung der Prozesswerte werden hierbei zwei *SystemUnitClasses* verwendet, eine um lesend auf einen Prozesswert eines Moduls zuzugreifen (*ProcessValueOut*) und einen um schreibend auf einen Prozesswert eines Moduls zuzugreifen (*ProcessValueIn*).

InternalElements der zwei genannten Klassen werden hierarchisch unterhalb der *Service InternalElements* angehängt. Je nachdem ob es ein *InternalElement* von *ProcessValueOut* oder von *ProcessValueIn* ist, verlinken die entsprechenden *InternalElements* auf ein lesendes oder schreibendes *InternalElement* von *DataAssembly*. Genau wie die anderen Parameter können Prozesswerte hierbei analog, digital oder binär sein. So ergibt sich folgende Zuordnung: *InternalElements* von *ProcessValueOut* verweisen entweder auf *AnaView*, *DigView* oder *BinView* *InternalElements* und *InternalElements* von *ProcessValueIn* verweisen *InternalElements* von einer der *SystemUnitClasses* *ExtAnaOp*, *ExtDigOp* oder *ExtBinOp*.

Die Gültigkeit des Prozesswerts wird vom Modul über den *WQC* signalisiert. Solange der *WQC* des Prozesswerts auf *Good* steht, ist der Prozesswert gültig. Wenn der Prozesswert nicht gültig ist, muss der *WQC* auf *Bad* gesetzt werden.

Die Umschaltung von der Nutzung eines externen Prozesswerts auf einen internen Prozesswert des Moduls kann z.B. durch einen Fahrweisenparameter oder Konfigurationsparameter realisiert werden.

2.2.5 Modellierung von Textlisten

Die Modellierung von Textlisten wird in der Dienstmodellierung für drei verschiedene Anwendungen verwendet:

1. Die Dienst-Bediener Interaktion wird über einen Frage-Antwort Katalog als Listen modelliert.
2. Der aktuelle Stand der Ausführung eines Dienstes kann über eine *Attribute* der Dienstschnittstelle auf einen Text verlinket werden.
3. Es besteht die Möglichkeit Enumerationen innerhalb von Dienstparametern zu verwenden um Bediener anstelle von Zahlen, Texte anzuzeigen

Abschnitt 2.2.1 zeigt die hierfür verwendeten *SystemUnitClasses* der Bibliothek *MTPTTextSUCLib*. Textlisten werden grundsätzlich in einer eigenen *InstanceHierarchy* angelegt. Diese wird über ein *InternalElement* von *TextSet*, auf die in [VDI/VDE/NAMUR 2658 Blatt 1] beschriebene Art und Weise im Inhaltsverzeichnis des Manifests zugänglich gemacht.

Innerhalb der *InstanceHierarchy* für Textlisten, werden alle Textlisten abgelegt. In Abhängigkeit des Typs der Textliste wird ein Wurzelement angelegt:

- Für Dienst-Bediener Interaktion ein *InternalElement* der *SystemUnitClass ServiceInteraction* angelegt
- Für den aktuellen Stand der Ausführung der Dienstes ein *InternalElement* der *SystemUnitClass ServicePosition*
- Für Enumerationen ein *InternalElement* der *SystemUnitClass EnumDefinition*

Jedes dieser *InternalElements* verlinket mit seinem *Attribute RefID* auf das *Attribute RefID* des zugehörigen Dienstes oder Parameters. Auf Grund der verwendeten *SystemUnitClass*, kann die Liste dem zugehörigen *Attribute* des zugehörigen *ServiceControl InternalElement* zugeordnet werden:

- Das *ServiceInteraction InternalElement* enthält die Fragen für *InteractQuestion* von *ServiceControl* und unterhalb der Fragen die Antworten für die Anzeige von *InteractAnswer*.
- Das *ServicePosition InternalElement* enthält die Liste der anzuzeigenden Texte für die Ausführung des Dienstes.
- Handelt es sich um ein *InternalElement* der *SystemUnitClass EnumDefinition*, so verweist dessen *Attribute RefID* auf den zugehörigen digitalen *ServiceParameter*.

Alle Listen werden grundsätzlich flach modelliert, der Name des *InternalElement* eines jeden Eintrags ist der Index innerhalb der Liste und wird für die zugehörige Ganzzahlvariable verwendet. Der Index muss nicht zwangsläufig hochgezählt werden, sondern muss lediglich innerhalb der Liste eindeutig und ganzzahlig sein. Der Name jeden Eintrags enthält damit den Wert der zugehörigen ganzzahligen Variablen. Die Ausnahme hiervon sind die Listen unterhalb von *ServiceInteraction*, welche unterhalb der Fragen wiederum eine weitere Liste der Antworten haben, welche in gleicher Art und Weise aufgebaut ist. Alle Listen werden inklusive des national language support nach [VDI/VDE/NAMUR 2658 Blatt 3] modelliert.

Ein Zusammenfassung ist in Abbildung 6 dargestellt.



































	Dienst-Bediener Interaktion	Anzeige des Verarbeitungsstands eines Dienstes	Enumeration eines Dienstparameters
<i>SystemUnitClasses</i>	 ServiceInteraction { Class LinkedObject }  Question { Class Text }  Answer { Class Text }	 ServicePosition { Class LinkedObject }  PositionText { Class Text }	 EnumDefinition { Class LinkedObject }  EnumItem { Class Text }
Text <i>InstanceHierarchy</i>	 ServiceInteraction { Class ServiceInteraction Role }  1 { Class Question Role }  1 { Class Answer Role }  2 { Class Answer Role }  2 { Class Question Role }  3 { Class Question Role }	 ServicePosition { Class ServicePosition Role }  1 { Class PositionText Role }  2 { Class PositionText Role }	 EnumDefinitionDigViewReportParameter1 { Class EnumDefinition }  1 { Class EnumItem Role }  2 { Class EnumItem Role }  8 { Class EnumItem Role }
Zugehöriges <i>InternalElement</i> der <i>InstanceList</i> und das verwendete <i>Attribute</i>	 InstanceList { Class InstanceList Role }  ServiceControl { Class ServiceControl Role }  InteractQuestion  InteractAnswer	 InstanceList { Class InstanceList Role }  ServiceControl { Class ServiceControl Role }  PostTextId	 DigViewReportParameter1 { Class DigView Role }  V
Zugehöriger Dienst oder Parameter in der Dienste <i>InstanceHierarchy</i>	 Services  Service1 { Class Service Role }	 Services  Service1 { Class Service Role }	 ReportParameter1 { Class ReportParameter Role }

Abbildung 6: Übersicht über die verwendeten Listenarten für die Dienste

2.2.6 Modellierung von Serviceabhängigkeiten

Noch im Modell zu definieren.

2.2.7 Anbindung an den Kommunikationsaspekt

Für die Modellierung wird das Konzept der *LinkedObjects*, wie in [VDI/VDE/NAMUR 2658 Blatt 1] beschrieben, verwendet. Die *InternalElements* der Dienstbeschreibung und das Manifest sind durch eine eindeutige Identifikation miteinander verbunden. Verbundene Elemente verwenden dafür das Attribut *RefId*. Eine Verbindung zwischen Objekten der Dienstbeschreibung und CDTs wird hergestellt,

wenn auf beiden Seiten die gleiche *RefId* verwendet wird, wodurch eine eindeutige Zuordnung gewährleistet ist.

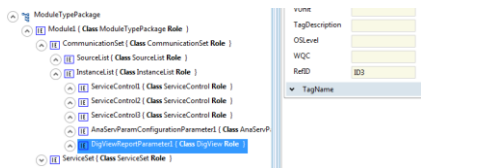
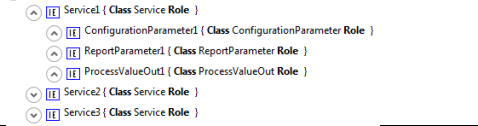
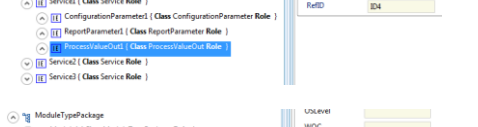
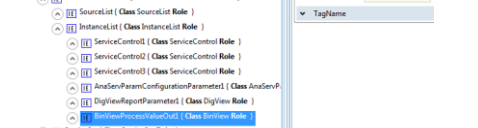
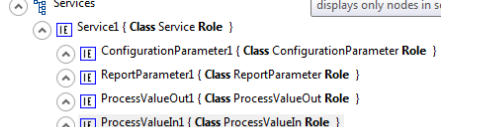
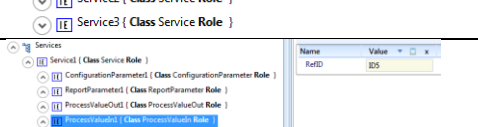
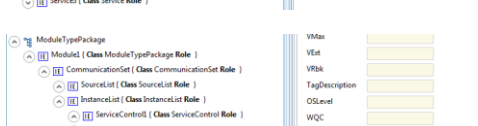
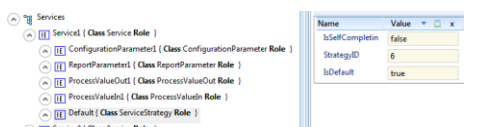
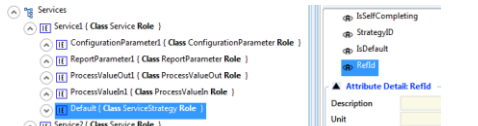
Für alle Objekte der Dienstbeschreibung gilt, dass nur Verweise auf Objekte erlaubt sind, die explizit in diesem Blatt der Richtlinie als weitere Ableitungsstufe von vorhandenen Objekten aus [VDI/VDE/NAMUR 2658 Blatt 3] definiert sind, Abbildung 4.

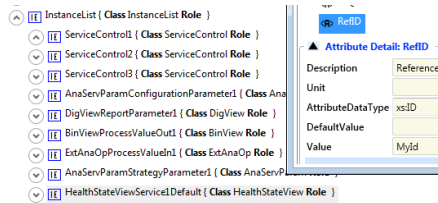
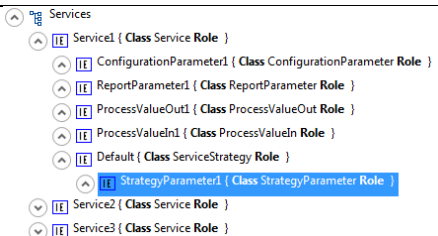
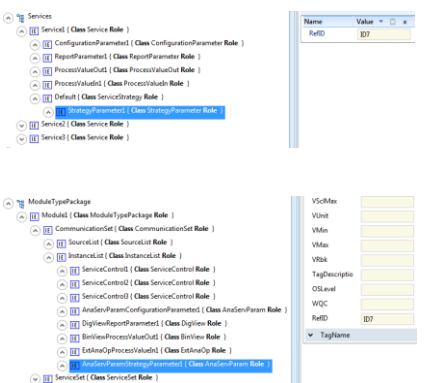
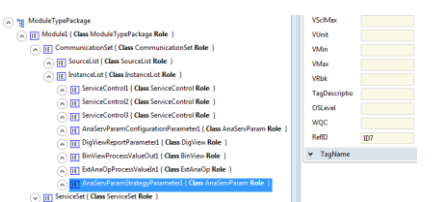

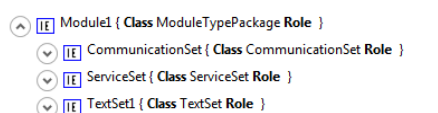
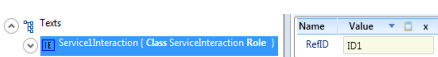

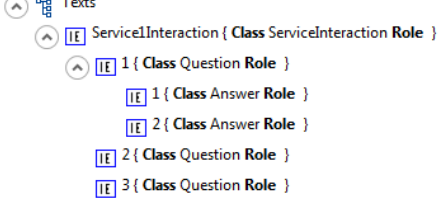
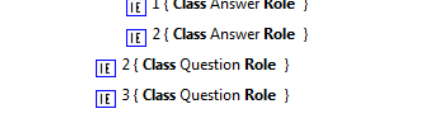
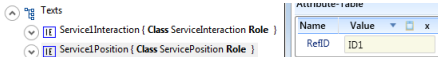
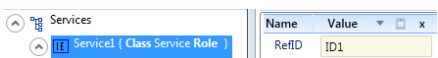
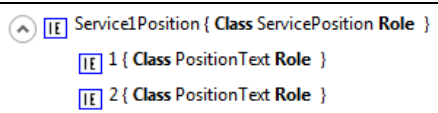
2.2.8 Modellierungsvorschriften

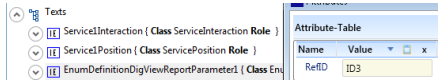
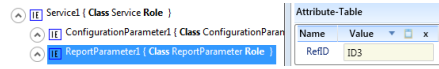
Tabelle 4 formuliert die in diesem Blatt festgelegte Spezifikation als handlungsorientierte Modellierungsvorschriften, die bei der Erstellung eines MTP eingehalten werden müssen.

Tabelle 4: Modellierungsvorschriften

ID	Regel	Verweis	Beispiel
1	Es gibt eine <i>InterfaceHierarchy</i> für alle Dienste eines Modules. Diese kann wie in [VDI/VDE/Namur 2658 Blatt 1] beschrieben entweder in der gleichen Datei wie das Manifest sein, oder in einer eigenen <i>AutomationML</i> Datei gespeichert werden.	[VDI/VDE/Namur 2658-1]	
2	Innerhalb der <i>InstanceHierarchy</i> wird für jeden Dienst ein <i>InternalElement</i> der <i>SystemUnitClass Service</i> angelegt. Der Name des <i>InternalElement</i> entspricht dem Namen des Dienstes.	2.2.1	
3	Im Manifest [VDI/VDE/Namur 2658 Blatt 1] wird hierarchisch unterhalb des <i>InternalElement</i> vom Typ <i>ModuleTypePackage</i> ein <i>InternalElement</i> der <i>SystemUnitClass ServiceSet</i> angelegt. Dieses verweist wie in [VDI/VDE/Namur 2658 Blatt 1] beschrieben auf die <i>InstanceHierarchy</i> der Dienste	[VDI/VDE/Namur 2658-1]	
4	Im Manifest [VDI/VDE/Namur 2658 Blatt 1] wird für jeden Dienst ein <i>InternalElement</i> der <i>SystemUnitClass ServiceControl</i> hierarchisch unterhalb des <i>InternalElements</i> der <i>SystemUnitClass InstanceList</i> angelegt.	2.2.2	
5	Das <i>InternalElement</i> der <i>SystemUnitClass Service</i> verweist mit seinem <i>Attribute RefId</i> auf <i>RefId</i> des zugehörigen <i>InternalElement</i> der <i>SystemUnitClass ServiceControl</i> im Manifest.	[VDI/VDE/Namur 2658-1]	
6	Jeder benötigte Konfigurationsparameter eines Dienstes wird als <i>InternalElement</i> der <i>SystemUnitClass ConfigurationParameter</i> hierarchisch unterhalb des Dienstes angehängt.	2.2.4.1	
7	Im Manifest [VDI/VDE/Namur 2658 Blatt 1] wird für jeden Konfigurationsparameter in Abhängigkeit dessen Typs ein <i>InternalElement</i> der <i>SystemUnitClasses AnaServParam, DigServParam</i> oder <i>BinServParam</i> hierarchisch unterhalb des <i>InternalElements</i> der <i>SystemUnitClass InstanceList</i> angelegt.	2.2.4.1	
8	Das <i>InternalElement</i> der <i>SystemUnitClass ConfigurationParameter</i> verweist mit seinem <i>Attribute RefId</i> auf <i>RefId</i> des zugehörigen <i>InternalElement</i> innerhalb der <i>InstanceList</i> .	2.2.4.1	
9	Jeder benötigte Reportparameter eines Dienstes wird als <i>InternalElement</i> der <i>SystemUnitClass ReportParameter</i> hierarchisch unterhalb des Dienstes angehängt.	2.2.4.3	
10	Im Manifest [VDI/VDE/Namur 2658 Blatt 1] wird für jeden Reportparameter in Abhängigkeit dessen Typs ein <i>InternalElement</i> der <i>SystemUnitClasses AnaView, DigView</i> oder <i>BinView</i> hierarchisch	2.2.4.3	

ID	Regel	Verweis	Beispiel
	unterhalb des <i>InternalElements</i> der <i>SystemUnitClass InstanceList</i> angelegt.		
11	Das <i>InternalElement</i> der <i>SystemUnitClass ReportParameter</i> verweist mit seinem <i>Attribute RefId</i> auf <i>RefId</i> des zugehörigen <i>InternalElement</i> innerhalb der <i>InstanceList</i> .	2.2.4.3	
12	Jeder ausgehende, für diesen Dienst relevante Prozesswert wird als <i>InternalElement</i> der <i>SystemUnitClass ProcessValueOut</i> hierarchisch unterhalb des Dienstes angehängt.	2.2.4.4	
13	Im Manifest [VDI/VDE/Namur 2658 Blatt 1] wird für jeden ausgehenden Prozesswert in Abhängigkeit dessen Typs ein <i>InternalElement</i> der <i>SystemUnitClasses AnaView, DigView</i> oder <i>BinView</i> hierarchisch unterhalb des <i>InternalElements</i> der <i>SystemUnitClass InstanceList</i> angelegt.	2.2.4.4	
14	Das <i>InternalElement</i> der <i>SystemUnitClass ProcessValueIn</i> verweist mit seinem <i>Attribute RefId</i> auf <i>RefId</i> des zugehörigen <i>InternalElement</i> innerhalb der <i>InstanceList</i> .	2.2.4.4	
15	Jeder eingehende, für diesen Dienst relevante Prozesswert wird als <i>InternalElement</i> der <i>SystemUnitClass ProcessValueIn</i> hierarchisch unterhalb des Dienstes angehängt.	2.2.4.4	
16	Im Manifest [VDI/VDE/Namur 2658 Blatt 1] wird für jeden eingehenden Prozesswert in Abhängigkeit dessen Typs ein <i>InternalElement</i> der <i>SystemUnitClasses ExtAnaOp, ExtDigOp</i> oder <i>ExtBinOp</i> hierarchisch unterhalb des <i>InternalElements</i> der <i>SystemUnitClass InstanceList</i> angelegt.	2.2.4.4	
17	Das <i>InternalElement</i> der <i>SystemUnitClass ProcessValueIn</i> verweist mit seinem <i>Attribute RefId</i> auf <i>RefId</i> des zugehörigen <i>InternalElement</i> innerhalb der <i>InstanceList</i> .	2.2.4.4	
18	Für jede Fahrweise des Dienstes wird ein <i>InternalElement</i> des <i>SystemUnitClass Strategy</i> hierarchisch unterhalb des Dienstes angehängt. Der Name des <i>InternalElements</i> entspricht der Name der Fahrweise.	2.2.3	
19	Jeder Dienst besitzt wenigstens eine Fahrweise, d.h. mindestens ein <i>InternalElement</i> der <i>SystemUnitClass ServiceStrategy</i> .	2.1.1	
20	<i>InternalElements</i> von <i>Strategy</i> haben die Attribute <i>IsSelfCompleting</i> , <i>StrategyId</i> und <i>IsDefault</i> . <i>IsSelfCompleting</i> wird auf wahr gesetzt, wenn sich der Dienst selbständig nach erfolgreicher Ausführung beendet. <i>IsDefault</i> wird für genau eine Fahrweise des Dienstes auf wahr gesetzt. Diese Fahrweise wird ausgeführt, falls die PFE keine andere Fahrweise für das Rezept vorgibt. <i>StrategyId</i> ist der Identifikationsmechanismus der Fahrweise und muss innerhalb eines Dienstes eindeutig einer Fahrweise zuweisbar sein.	2.2.4.2	
21	Im Manifest [VDI/VDE/Namur 2658 Blatt 1] wird für jede Fahrweise ein <i>InternalElement</i> der <i>SystemUnitClasses HealthStateView</i> hierarchisch unterhalb des <i>InternalElements</i> der <i>SystemUnitClass InstanceList</i> angelegt.	2.1.4	
22	Das <i>InternalElement</i> der <i>SystemUnitClass Strategy</i> verweist mit seinem <i>Attribute RefId</i> auf	2.1.4	

ID	Regel	Verweis	Beispiel
	<i>RefId</i> des zugehörigen <i>InternalElement</i> innerhalb der <i>InstanceList</i> .		
22	Jeder für diese Fahrweise relevante Fahrweisenparameter wird als <i>InternalElement</i> der <i>SystemUnitClass StrategyParameter</i> hierarchisch unterhalb der Fahrweise angehängt.	2.2.4.2	
23	Im Manifest [VDI/VDE/Namur 2658 Blatt 1] wird für jeden Fahrweisenparameter in Abhängigkeit dessen Typs ein <i>InternalElement</i> der <i>SystemUnitClasses</i> <i>AnaServParam</i> , <i>DigServParam</i> oder <i>BinServParam</i> hierarchisch unterhalb des <i>InternalElements</i> der <i>SystemUnitClass InstanceList</i> angelegt.	2.2.4.2	
24	Das <i>InternalElement</i> der <i>SystemUnitClass StrategyParameter</i> verweist mit seinem <i>Attribute RefId</i> auf <i>RefId</i> des zugehörigen <i>InternalElement</i> innerhalb der <i>InstanceList</i> .	2.2.4.2	
25	Werden Textlisten für die Dienste benötigt, so wird eine <i>InstanceHierarchy</i> innerhalb der MTP für diesen Aspekt angelegt.	2.2.5	
26	Im Manifest [VDI/VDE/Namur 2658 Blatt 1] wird hierarchisch unterhalb des <i>InternalElement</i> vom Typ <i>ModuleTypePackage</i> ein <i>InternalElement</i> der <i>SystemUnitClass TextSet</i> angelegt. Dieses verweist wie in [VDI/VDE/Namur 2658 Blatt 1] beschrieben auf die <i>InstanceHierarchy</i> der Dienste	2.2.5	
27	Für jeden Dienst der eine Dienst-Bediener Interaktion benötigt wird innerhalb des Text Aspekts ein <i>InternalElement</i> der <i>SystemUnitClass ServiceInteraction</i> angelegt.	2.2.5	
28	Das <i>InternalElement</i> der <i>SystemUnitClass ServiceInteraction</i> verweist mit seinem <i>Attribute RefId</i> auf den zugehörigen Dienst in der Dienste <i>InstanceHierarchy</i> .	2.2.5	
29	Innerhalb der <i>InternalElement</i> der <i>SystemUnitClass ServiceInteraction</i> , wird die Liste der Frage als <i>InternalElements</i> der <i>SystemUnitClass Question</i> angelegt.	2.2.5	
30	Innerhalb der <i>InternalElements</i> der <i>SystemUnitClass Question</i> , wird die Liste der Fragen als <i>InternalElements</i> der <i>SystemUnitClass Answer</i> angelegt.	2.2.5	
31	Für jeden Dienst der seinen Ausführungsstatus als Text anzeigen soll, wird innerhalb des Text Aspekts ein <i>InternalElement</i> der <i>SystemUnitClass ServicePosition</i> angelegt.	2.2.5	
32	Das <i>InternalElement</i> der <i>SystemUnitClass ServicePosition</i> verweist mit seinem <i>Attribute RefId</i> auf den zugehörigen Dienst in der Dienste <i>InstanceHierarchy</i> .	2.2.5	
33	Innerhalb der <i>InternalElement</i> der <i>SystemUnitClass ServicePosition</i> , wird die Liste der Rückmeldungen zur Ausführung als <i>InternalElements</i> der <i>SystemUnitClass PositionText</i> angelegt.	2.2.5	

ID	Regel	Verweis	Beispiel
34	Für jeden Dienstparameter der als Enumeration verwendet werden soll, wird innerhalb des Text Aspekts ein <i>InternalElement</i> der <i>SystemUnitClass EnumDefinition</i> angelegt.	2.2.5	
35	Das <i>InternalElement</i> der <i>SystemUnitClass EnumDefinition</i> verweist mit seinem <i>Attribute RefID</i> auf den zugehörigen Dienstparameter in der Dienste <i>InstanceHierarchy</i> .	2.2.5	
36	Innerhalb der <i>InternalElement</i> der <i>SystemUnitClass EnumDefinition</i> , wird die Liste der Rückmeldungen zur Ausführung als <i>InternalElements</i> der <i>SystemUnitClass EnumItem</i> angelegt.	2.2.5	