



Hensel, Bloch, Menschner

# **BEGLEITDOKUMENTATION ZUR SERVICEBESCHREIBUNG INNERHALB DES MTP**

*VERSION 0.1*




# INHALT

1	Dokumentenhistorie .....	4
2	Kontakte.....	4
3	UML Beschreibung .....	5
3.1	Service.....	5
3.1.1	Elternklasse .....	5
3.1.2	Attribute .....	5
3.1.3	Ausprägungen .....	6
3.1.4	Relationen zu anderen Klassen .....	6
3.2	ServiceParameter.....	6
3.2.1	Elternklasse .....	6
3.2.2	Attribute .....	6
3.2.3	Ausprägungen .....	7
3.2.4	Relationen zu anderen Klassen .....	7
3.3	ServiceStateLoop.....	7
3.3.1	Elternklasse .....	7
3.3.2	Attribute .....	7
3.3.3	Ausprägungen .....	7
3.3.4	Relationen zu anderen Klassen .....	7
3.4	ServiceState .....	7
3.4.1	Elternklasse .....	7
3.4.2	Attribute .....	8
3.4.3	Ausprägungen .....	8
3.4.4	Relationen zu anderen Klassen .....	8
3.5	ServiceTransition .....	8
3.5.1	Elternklasse .....	8
3.5.2	Attribute .....	8
3.5.3	Ausprägungen .....	8
3.5.4	Relationen zu anderen Klassen .....	8
3.6	ServiceRelationMatrix .....	9

Name	Begleitdokumentation zur Servicebeschreibung
Ersteller	Hensel, Bloch, Menschner
Institutionen	Technische Universität Dresden, Helmut-Schmidt-Universität Hamburg,
Letzte Bearbeitung	06.09.2017
Version	0.1
Copyright	

3.6.1	Elternklasse .....	9
3.6.2	Attribute .....	9
3.6.3	Ausprägungen .....	9
3.6.4	Relationen zu anderen Klassen .....	9
3.7	ServiceRelation .....	9
3.7.1	Elternklasse .....	9
3.7.2	Attribute .....	9
3.7.3	Ausprägungen .....	9
3.7.4	Relationen zu anderen Klassen .....	10

Name	Begleitdokumentation zur Servicebeschreibung
Ersteller	Hensel, Bloch, Menschner
Institutionen	Technische Universität Dresden, Helmut-Schmidt-Universität Hamburg,
Letzte Bearbeitung	06.09.2017
Version	0.1
Copyright	

# 1 DOKUMENTENHISTORIE

Version	Datum	Autor	Kommentar
0.1	2017-09-06	Hensel	Erster Draft
0.2	2017-09-12	Menschner	
0.3	2017-09-12	Bloch	

# 2 KONTAKTE

Stephan Hensel (TU Dresden), [stephan.hensel@tu-dresden.de](mailto:stephan.hensel@tu-dresden.de)

Henry Bloch (HSU Hamburg), [henry.bloch@hsu-hh.de](mailto:henry.bloch@hsu-hh.de)

Anna Menschner (TU Dresden) [anna.hahn@tu-dresden.de](mailto:anna.hahn@tu-dresden.de)

Leon Urbas (TU Dresden), [leon.urbas@tu-dresden.de](mailto:leon.urbas@tu-dresden.de)

Alexander Fay (HSU Hamburg), [alexander.fay@hsu-hh.de](mailto:alexander.fay@hsu-hh.de)

Name	Begleitdokumentation zur Servicebeschreibung
Ersteller	Hensel, Bloch, Menschner
Institutionen	Technische Universität Dresden, Helmut-Schmidt-Universität Hamburg,
Letzte Bearbeitung	06.09.2017
Version	0.1
Copyright	

## 3 UML BESCHREIBUNG

Nachfolgend wird eine kurze Beschreibung zum UML Modell der Servicebeschreibung innerhalb des MTP gegeben. Das UML Modell beschreibt hierbei die Abbildungsvorschrift der Services im Module Type Package (MTP), die später auf die technologische Umsetzung mit AutomationML gemappt werden soll. Die einzelnen vorhandenen Objekte werden direkt als System Unit Classes im UML beschrieben. Auf die Angabe von Attributen, die in einer AutomationML-Umsetzung zwingend erforderlich sind wird überwiegend verzichtet. In der Beschreibung sind in UML verwendete Bezeichnungen kursiv hervorgehoben.

### 3.1 SERVICE

Durch Services (*Service*) werden verfahrenstechnische Funktionen in der modularen Prozessautomation gekapselt. In der Prozessführungsebene (PFE) werden die Services dann so orchestriert, dass der gewünschte Produktionsablauf realisiert wird

#### 3.1.1 Elternklasse

Ein *Service* ist von einem *LinkedObject* abgeleitet.

#### 3.1.2 Attribute

Jeder Service wird durch mehrere Attribute bzw. Variablen gekennzeichnet. Hierzu gehören ein Name (*Name*), die Betriebsart (*OperationMode*), der Status (*Status*), das Kommando (*Command*), der Status der Parameter (*ParameterStatus*) und das Parameterkommando (*ParameterCommand*). Dem Ansatz von PackML folgend wird für jede Betriebsart eines Service eine neue Instanz des Zustandsautomaten modelliert, dies ist über unterschiedliche Angaben in der Betriebsart des Service anzugeben. Die zu modellierenden Betriebsarten sind innerhalb der Arbeitskreise gesondert zu diskutieren und festzusetzen. Hierbei empfiehlt es sich ausgehend von bestehenden Standards wie FoundationFieldbus oder Profibus ausgehend die innerhalb der modularen Prozessautomation notwendigen Betriebsarten für Services zu analysieren. Status und Kommando für einen Service folgen jeweils den Definitionen des innerhalb der Arbeitskreise festgelegten Zustandsautomaten, der in Abbildung 1 dargestellt ist. Mittels des Status kann der aktuelle Zustand des Service abgefragt werden. Es wird der im Zustandsmodell festgelegte numerische Wert zurückgegeben, in Analogie zu PackML. Mittels des Kommandos können Zustandsübergänge eingeleitet werden. Hierbei können die numerischen Werte übergeben werden, die im Zustandsautomaten Transitionen zugeordnet sind.. Durch ein Parameterkommando wird der Service dazu aufgefordert die aktuell angelegten Parameter zu übernehmen. Durch den Parameterstatus wird angegeben, ob die Parameter erfolgreich übernommen wurden. Hierzu werden die Parameter innerhalb der PFE vorgehalten und immer als gesamtes Paket in das Modul geschrieben.

Name	Begleitdokumentation zur Servicebeschreibung
Ersteller	Hensel, Bloch, Menschner
Institutionen	Technische Universität Dresden, Helmut-Schmidt-Universität Hamburg,
Letzte Bearbeitung	06.09.2017
Version	0.1
Copyright	

### 3.1.3 Ausprägungen

Services können zwei Ausprägungen besitzen. Entweder handelt es sich um einen kontinuierlichen Service (*ContinuousService*), dieser muss durch einen expliziten Befehl abgebrochen werden, oder es handelt sich um einen selbstbeendenden Service (*SelfcompletingService*), der automatisch nach der Bearbeitung in Complete übergeht. Der wesentliche Unterschied zwischen den beiden Servicetypen ist der Zustand Complete. Ist dieser Zustand vorhanden (*SelfcompletingService*), so wird innerhalb des Service der Zustandsübergang von Running in Complete eigenständig durchgeführt (ohne direkten Einfluss durch die PFE). Ein *ContinuousService* wiederum verbleibt nach dem Eintritt in den Zustand Running so lange in diesem Zustand bis ein weiteres Kommando von außerhalb des Services gesetzt wird, dass eine Zustandsübergang von Running ausgehend hervorruft.

### 3.1.4 Relationen zu anderen Klassen

Für jeden Service müssen die Zustände Idle, Running und Aborted implementiert werden. Des Weiteren muss immer die Abort-Loop vorhanden sein, um den Service im Notfall abubrechen. Darüber hinaus hat ein selbstbeendender Service (*SelfcompletingService*) auch immer einen Complete Zustand, der anzeigt, dass die Bearbeitung abgeschlossen ist. Ein kontinuierlicher Service hat diesen Zustand nicht, da dieser durch Aufruf einer Transition beendet werden muss und damit den Complete Zustand nie erreichen würde. Alle anderen Schleifen, wie Pause, Hold und Stop sind optional und müssen nicht implementiert werden. Wenn Schleifen implementiert werden, so müssen immer alle Zustände der Schleife implementiert werden.

Ein Service kann des Weiteren eine beliebige Anzahl an Serviceparametern besitzen.

Die Ausgestaltung des Zustandsautomaten eines jeden Service und dessen Serviceparametern obliegt dem Modulbauer.

## 3.2 SERVICEPARAMETER

Mittels der Parameter eines Service (*ServiceParameter*) können Einstellungen an diesem Service vorgenommen werden. Beispielsweise kann die Dauer eines Dosierprozesses eines selbstbeendenden Service oder die Durchflussrate eines kontinuierlichen Service angegeben werden.

Die Auswertung der jeweiligen Parameter erfolgt immer, wenn ein entsprechender Schreibbefehl an das Modul gesendet wird (vgl. 3.1.2).

### 3.2.1 Elternklasse

Ein *ServiceParameter* ist von einem *LinkedObject* abgeleitet.

### 3.2.2 Attribute

Jedem Serviceparameter sind ein Name (*Name*), eine Beschreibung (*Description*), eine Einheit (*Unit*), ein Standard-Wert (*DefaultValue*) und ein aktueller Wert (*Value*) als Attribute zugewiesen. Der aktuelle Wert verweist dabei beispielsweise auf eine Variable in einem OPC UA Server von der der aktuelle Wert ausgelesen werden kann.

Name	Begleitdokumentation zur Servicebeschreibung
Ersteller	Hensel, Bloch, Menschner
Institutionen	Technische Universität Dresden, Helmut-Schmidt-Universität Hamburg,
Letzte Bearbeitung	06.09.2017
Version	0.1
Copyright	

### 3.2.3 Ausprägungen

Es werden keine Serviceparametertypen unterschieden.

### 3.2.4 Relationen zu anderen Klassen

Serviceparameter sind Services zugeordnet. Des Weiteren können zu einem Serviceparameter die Zustände angegeben werden (*correspondingStates*), in denen eine Auswertung des Parameters stattfindet. Eine Angabe ist jedoch nicht zwingend und dementsprechend nur als optional anzusehen.

## 3.3 SERVICESTATELOOP

Der Zustandsautomation in Abbildung 1 ist in mehrere Schleifen untergliedert, die jeweils aus einzelnen Zuständen bestehen. Da Schleifen nur als Ganzes implementiert werden oder weggelassen werden dürfen, muss eine Beschreibung im MTP erfolgen, die die verfügbaren Schleifen angibt.

### 3.3.1 Elternklasse

Eine *ServiceStateLoop* ist von einem *LinkedObject* abgeleitet.

### 3.3.2 Attribute

Es sind keine weiteren Attribute zugeordnet.

### 3.3.3 Ausprägungen

Es findet eine Spezialisierung in die im Zustandsautomaten definierten Schleifen statt. Hierzu zählen Stop (*StopServiceStateLoop*), Hold (*HoldServiceStateLoop*), Pause (*PauseServiceStateLoop*) und Abort (*AbortServiceStateLoop*).

### 3.3.4 Relationen zu anderen Klassen

Die verfügbaren Schleifen innerhalb eines Services sind einem Serviceobjekt zugeordnet. Hierüber erfolgt auch die Beschreibung, ob eine Schleife implementiert werden muss oder ob diese optional ist. Jeder Schleife sind die Zustände zugeordnet, die in ihr implementiert werden müssen.

## 3.4 SERVICESTATE

Die im UML beschriebenen Zustände repräsentieren die Zustände des Zustandsautomaten nach Abbildung 1.

### 3.4.1 Elternklasse

Ein *ServiceState* ist von einem *LinkedObject* abgeleitet.

Name	Begleitdokumentation zur Servicebeschreibung
Ersteller	Hensel, Bloch, Menschner
Institutionen	Technische Universität Dresden, Helmut-Schmidt-Universität Hamburg,
Letzte Bearbeitung	06.09.2017
Version	0.1
Copyright	

### 3.4.2 Attribute

Es sind keine weiteren Attribute zugeordnet.

### 3.4.3 Ausprägungen

Es findet eine Spezialisierung in die im Zustandsautomaten angegebenen Zustände statt. Hierzu zählen Idle (*IdleState*), Running (*RunningState*), Aborted (*AbortedState*), Aborting (*AbortingState*), Complete (*CompleteState*), Paused (*PausedState*), Pausing (*PausingState*), Holding (*HoldingState*), Held (*HeldState*), Unholding or Restarting (*UnholdingOrRestartingState*), Stopped (*StoppedState*) und Stopping (*StoppingState*) statt.

### 3.4.4 Relationen zu anderen Klassen

Die Zustände werden den entsprechenden Schleifen bzw. dem Service direkt zugeordnet und optional durch die Serviceparameter referenziert. Einem Zustand kann weiterhin optional das Equipment zugewiesen werden, das in dem Servicezustand Verwendung findet (*occupiedEquipment*). Hierbei wird auf das bestehende *DataAssembly* verwiesen, das auch für das HMI verwendet wird. Dadurch kann eine Liste mit allem im Zustand verwendeten Equipment ausgegeben werden. Des Weiteren werden die Zustände in den Servicereaktionen entweder als *source* oder *target* Element verwendet. Servicetransitionen ist jeweils genau ein *source* und ein *target* Zustand zugeordnet.

## 3.5 SERVICETRANSITION

Die im UML beschriebenen Transitionen repräsentieren die Transitionen des Zustandsautomaten nach Abbildung 1.

### 3.5.1 Elternklasse

Eine *ServiceTransition* ist von einem *LinkedObject* abgeleitet.

### 3.5.2 Attribute

Jeder Servicetransition ist ein Name (*Name*) zugewiesen. Dieser kennzeichnet die entsprechende Transition.

### 3.5.3 Ausprägungen

Es findet keine Spezialisierung statt.

### 3.5.4 Relationen zu anderen Klassen

Servicetransitionen ist jeweils genau ein *source* und ein *target* Zustand zugeordnet. Hierdurch werden die Richtung und die zugehörigen Zustände der Transition gekennzeichnet. Des Weiteren werden die Transitionen in den Servicereaktionen entweder als *source* und/oder *target* Element verwendet.

Name	Begleitdokumentation zur Servicebeschreibung
Ersteller	Hensel, Bloch, Menschner
Institutionen	Technische Universität Dresden, Helmut-Schmidt-Universität Hamburg,
Letzte Bearbeitung	06.09.2017
Version	0.1
Copyright	



## 3.6 SERVICERELATIONMATRIX

Die Matrix der Servicereaktionen (*ServiceRelationMatrix*) beschreibt alle vorhandenen Servicereaktionen zwischen den unterschiedlichen Services des Moduls.

### 3.6.1 Elternklasse

Eine *ServiceRelationMatrix* ist von einem *LinkedObject* abgeleitet.

### 3.6.2 Attribute

Es sind keine weiteren Attribute zugeordnet.

### 3.6.3 Ausprägungen

Es findet keine Spezialisierung statt.

### 3.6.4 Relationen zu anderen Klassen

Die Matrix der Servicereaktionen kann beliebig viele Servicereaktionen zwischen unterschiedlichen Services des Modus beinhalten.

## 3.7 SERVICERELATION

Servicereaktionen (*ServiceRelation*) beschreiben Relationen/Abhängigkeiten zwischen unterschiedlichen Services eines Moduls.

### 3.7.1 Elternklasse

Eine *ServiceRelation* ist von einem *LinkedObject* abgeleitet.

### 3.7.2 Attribute

Es sind keine weiteren Attribute zugeordnet.

### 3.7.3 Ausprägungen

Es findet eine Spezialisierung in drei Klassen statt, die jeweils unterschiedliche Relationsarten beschreiben. Hierzu gehören die Sperrung von Transitionen auf Basis von Zuständen (*DisableServiceRelation*), das Erlauben von Transitionen auf Basis von Zuständen (*EnableServiceRelation*) und das synchrone Weiterschalten von Transitionen (*SyncServiceRelation*).

Beispiele dazu wären:

- *DisableServiceRelation*
  - *Source: Service1 .Running; Target: Service2.Start*
  - *-> Service 2 kann nicht gestartet werden, solange Service 1 im Zustand Running ist*
- *EnableServiceRelation*

Name	Begleitdokumentation zur Servicebeschreibung
Ersteller	Hensel, Bloch, Menschner
Institutionen	Technische Universität Dresden, Helmut-Schmidt-Universität Hamburg,
Letzte Bearbeitung	06.09.2017
Version	0.1
Copyright	

- *Source: Service1 .Running; Target: Service2.Start*
- Nur wenn Service 1 sich im Zustand *Running* befindet, ist die Transition *Start* im Service 2 schaltfähig.
- *SyncServiceRelation*
  - *Source: Service1.Start; Target: Service2.Start*
  - Wenn Service 1 gestartet wird, muss auch S2 gestartet werden

Sind für bestimmte Kombinationen keine Relationen/ Abhängigkeiten angegeben, so besteht kein funktionaler oder sicherheitsrelevanter Zusammenhang.

### 3.7.4 Relationen zu anderen Klassen

Die einzelnen Ausprägungen haben Beziehungen zu Servicetransitionen bzw. Servicezuständen. Diese werden entweder als *source* oder *target* verwendet. Wenn mehrere Elemente in *source* bzw. *target* angegeben werden, so sind dieser jeweils durch ein logisches UND miteinander verbunden. Eine ODER-Verknüpfung ist durch das Anlegen von verschiedenen Relationen des gleichen Typs mit unterschiedlichen *source/target*-Elementen möglich. Das *source*-Element beschreibt jeweils die Elemente, die für das Auslösen einer Relation verantwortlich sind. Das *target*-Element beschreibt dementsprechend die Elemente, die von der Relation betroffen sind. Enable und Disable Relationen verwenden dabei als Ursache stets Zustände und als Wirkung das exklusive Erlauben oder Verriegeln von Transitionen. Demgegenüber verwendet die Synchronisationsrelation Transitionen als Ursache und Wirkung. Das bedeutet, wenn die verursachenden Transitionen ausgelöst werden, so müssen auch stets die Transitionen des *target*-Elementes ausgelöst werden.

Name	Begleitdokumentation zur Servicebeschreibung
Ersteller	Hensel, Bloch, Menschner
Institutionen	Technische Universität Dresden, Helmut-Schmidt-Universität Hamburg,
Letzte Bearbeitung	06.09.2017
Version	0.1
Copyright	

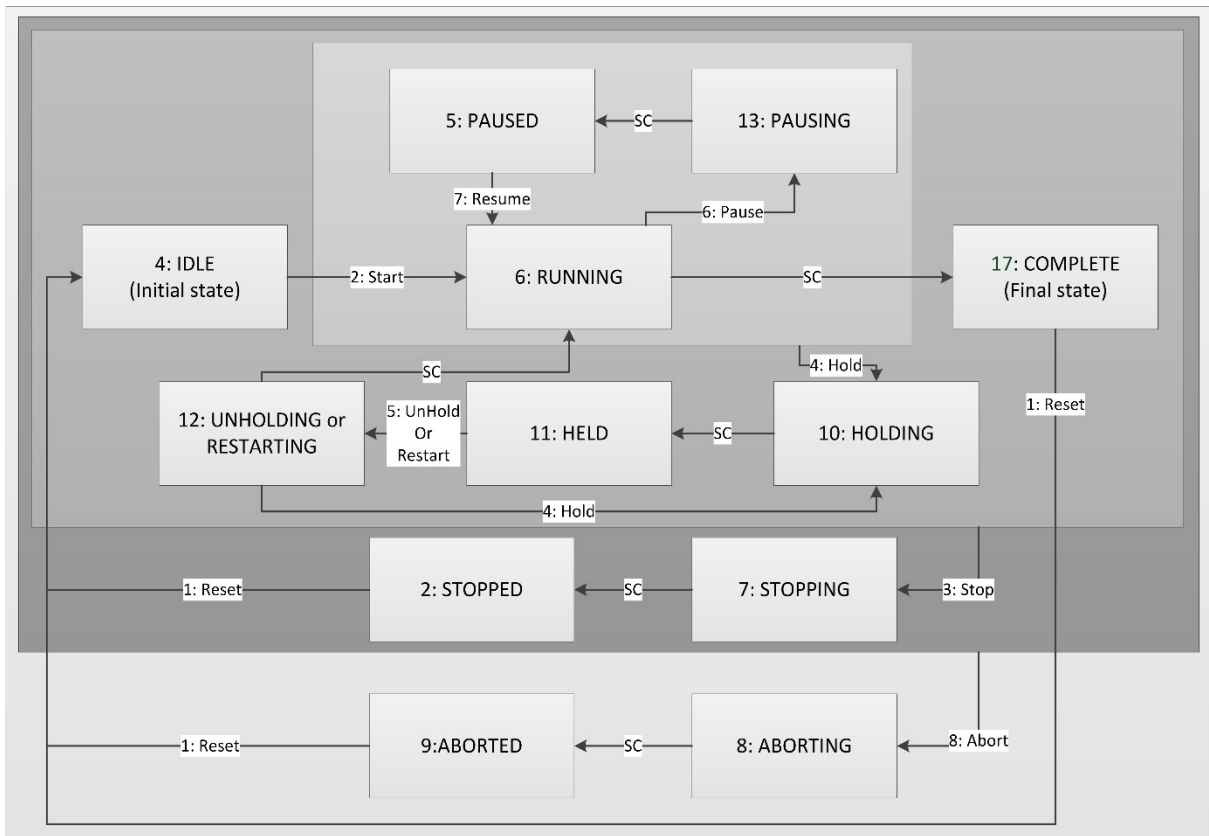


Abbildung 1 - Zustandsautomat

Name	Begleitdokumentation zur Servicebeschreibung
Ersteller	Hensel, Bloch, Menschner
Institutionen	Technische Universität Dresden, Helmut-Schmidt-Universität Hamburg,
Letzte Bearbeitung	06.09.2017
Version	0.1
Copyright	██████