Funktionsbausteine und SSCs

Modellaggregation und Handhabungsguideline

Dipl.-Ing. Lars Evertz

16.03.2016

Lehrstuhl für Prozessleittechnik
Prof. Dr.-Ing. Ulrich Epple
RWTH Aachen
D-52064 Aachen, Deutschland
Telefon +49 241 80 94339
Fax +49 241 80 92238
www.plt.rwth-aachen.de

Inhalt

1 Übersicht

Dieses Dokument beschreibt zunächst den Umgang mit Funktionsbausteinen (Entwicklung und Engineering) und darauf aufbauend mit Sequential State Charts zur Ablaufmodellierung. Anschließend wird darauf eingegangen, wie die besagten Programmmodelle zum Aufbau von Gruppensteuerungen (Sonderfunktionen) und Rezeptabläufen eingesetzt werden können. Die Anwendungsmöglichkeiten werden anhand zweier Beispiele vorgestellt. Diese Beispiele (ein kontinuierlicher Regler und ein Rezeptablauf) sind als in sich geschlossene funktionale Einheiten umgesetzt, die nicht mit dem Prozess „verdrahtet“ werden, sondern über definierte Schnittstellen auf die Einzelsteuerungen einwirken und Systemzustände abfragen.

2 Funktionsbausteine

Funktionsbausteine stellen den größten Teil der verwendeten aktiven Instanzen dar. Sie zeichnen sich durch zyklischen Aufruf einer internen Methode aus, die in der Lage ist, beliebige Funktionalitäten umzusetzen. Wie alle anderen Modelle in der acplt-Laufzeitumgebung, werden Funktionsbausteine zunächst als Klassen definiert und ihre Funktionalität in C ausimplementiert (Entwicklungsphase). Anschließend können sie in das acplt-Laufzeitsystem geladen und instanziiert werden (Engineeringphase). Die beiden Phasen werden nun beschrieben.

2.1 Entwicklung von Funktionsbausteinen

Die Entwicklung eines Funktionsbausteins beginnt mit der Definition der Klasse. Eine Funktionsbausteinklasse muss unmittelbar oder mittelbar von der Klasse *functionblock* der Bibliothek *fb* abgeleitet sein. Durch diese Ableitung werden die entsprechenden Tasking-Eigenschaften erzeugt und die neue Klasse als Funktionsbaustein aufrufbar.

Funktionsbausteine enthalten Variablen, die unterschiedliche Zugriffstypen haben können. Diese Typen werden durch Setzen der semantischen Flags der Variablen festgelegt. Es gibt Eingänge, Ausgänge, Parameter, interne und versteckte Variablen. Diese Variablentypen unterscheiden sich hinsichtlich ihrer Zugriffsrechte. Eingänge können mit anderen Funktionsbausteinen verbunden werden. Dabei ist lesender und schreibender Zugriff erlaubt. Ausgänge werden nur vom Funktionsbaustein selbst beschrieben. Sie können Quelle von Verbindungen sein. Parameter sind Eingänge, die nicht mit anderen Funktionsbausteinen verbunden werden können. Sie sind zur Konfiguration eines Funktionsbausteins gedacht. Interne Variablen können nur lesend zugegriffen werden. Sie können nicht als Quelle von Verbindungen dienen. Versteckte Variablen können ebenfalls nur lesend zugegriffen und nicht verbunden werden. Der Begriff versteckt bezeichnet dabei keine echte Auswirkung auf die variable, sondern dient lediglich dazu, einem Anzeigeprogramm einen Darstellungshinweis zu geben.

Die Definition einer Funktionsbausteinklasse beinhaltet die Definition der zum Baustein gehörenden Variablen und deren Typs und Zugriffstyps. Außerdem wird definiert, ob die Klasse spezielle Lebenszyklusfunktionen (constructor, destructor, startup, shutdown, checkinit) implementiert. Werden diese Funktionen nicht definiert, so wird automatisch die entsprechende Funktion der basisklasse verwendet. Neben diesen Funktionen hat jeder Funktionsbaustein eine so genannte typemethod. Diese setzt die Funktionalität des Funktionsbausteins um. Daher muss sie für jeden Funktionsbaustein definiert werden. Des Weiteren können beliebig weitere Funktionen definiert werden, die der Funktionsbaustein bereitstellt. Letztere werden jedoch nicht zyklisch ausgeführt. Allein die typemethod wird über das Tasking-System getriggert. Listing 1 zeigt beispielhaft die Definition der Funktionsbausteinklasse *ADD* der Bibliothek *iec61131stdfb*. Wie dargestellt können sowohl die Klasse selbst als auch jede Variable einen Kommentar und semantische Flags tragen. Der Ausdruck IS_INSTANTIABLE; legt fest, dass diese klasse instanziiert werden kann. Die Klasse verfügt über zwei Eingänge und einen Ausgang vom Variablentyp ANY. Sie stellt neben der typemethod auch einen eigenen constructor und destructor bereit. Die besagten Funktionen werden vom Entwickler in C ausimplementiert. Dabei ist darauf zu achten, dass es nicht zu langen Laufzeiten der Funktionen kommen darf, da das Tasking-System rein kooperativ arbeitet.

```
1  CLASS ADD : CLASS fb/functionblock
2      IS_INSTANTIABLE;
3      COMMENT = "Addition";
4      VARIABLES
5        IN1 : ANY                          HAS_SET_ACCESSOR
6              FLAGS = "i";
7        IN2 : ANY                          HAS_SET_ACCESSOR
8              FLAGS = "i";
9        OUT : ANY                          HAS_GET_ACCESSOR
10             FLAGS = "o"
11             COMMENT = "OUT = IN1 + IN2 + ...";
12     END_VARIABLES;
13 OPERATIONS
14       destructor : C_FUNCTION <OV_FNC_DESTRUCTOR>;
15       constructor     : C_FUNCTION <OV_FNC_CONSTRUCTOR>;
16       typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
17 END_OPERATIONS;
18  END_CLASS;
```

Listing 1: Definition der Funktionsbausteinklasse ADD

Ist eine oder mehrere Funktionsbausteinklassen fertig implementiert, so kann daraus eine Bibliothek erzeugt werden. Die erzeugte Bibliothek kann nun zur Laufzeit in einen RTE-Server geladen werden. Damit sind dem Server die in der Bibliothek enthaltenen Klassen bekannt und können nun instanziiert werden.

2.2 Umgang mit Funktionsbausteinen im Laufzeitsystem

Der Umgang mit Funktionsbausteinen sieht immer folgendermaßen aus:

- Laden der gewünschten Klassenbibliotheken
- Erzeugen einer Funktionsbausteininstanz
- Einordnen der Instanz in die gewünschte Taskliste (erledigen Engineering-Werkzeuge teilweise automatisch)
- Setzen der Parametereingänge und eventuell der Variablentypen der ANY-Variablen
- Anlagen der Verbindungen zu anderen Funktionsbausteinen
- Aktivieren des Bausteins

Auf diese Punkte wird nun näher eingegangen. Dabei wird als Engineering-Werkzeug das cshmi fb-Engineering verwendet. Andere Werkzeuge sind im Umgang anders, führen aber die gleichen Aktionen durch.

Aus der Liste der auf dem Server verfügbaren Bibliotheken im oberen Teil der Engineering Sicht (vergleiche Abbildung 1) wird die Bibliothek gewählt, aus der eine Instanz angelegt werden soll. Durch einen Klick auf die gewünschte Bibliothek (in diesem Beispiel *iec61131stdfb*) öffnet sich eine Liste der Klassen dieser Bibliothek, wie in Abbildung 2 dargestellt.

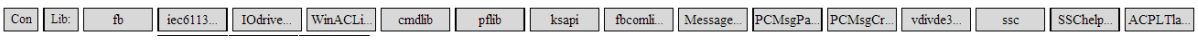

Abbildung 1: Liste der verfügbaren Bibliotheken (Ausnahmen: Con - Erzeugt eien neue Verbindung; Lib - lädt eine neue Klassenbibliothek)

Ist die gewünschte Bibliothek nicht geladen, so kann dies durch einen Klick auf Lib erfolgen. Dabei öffnet sich ein Fenster zur Eingabe des Namens der gewünschten Bibliothek.

Abbildung 2: Liste der Klassen in der Bibliothek *iec61131stdfb*

Nach einem Klick auf die gewünschte Klasse (hier *ADD*) öffnet sich ein Fenster mit der Aufforderung zur Eingabe des Namens der Instanz (Abbildung 3). Nach Eingabe des Namens und Bestätigung wird die neue Instanz angelegt (Abbildung 4). Funktionsbausteine werden von diesem Werkzeug automatisch in eine Taskliste eingehängt. Diese Liste ist in der zweiten Zeile des Funktionsbausteinkopfes angegeben. Funktionsbausteine werden entweder dem UrTask (hier im Beispiel) oder, wenn sie in einem Funktionsblatt instanziiert wurden, der Taskliste des Funktionsblatts zugeordnet. Die Ausführung des Funktionsbausteins ist zunächst deaktiviert. Dies ist an der gelben Färbung des Kopfes sichtbar.

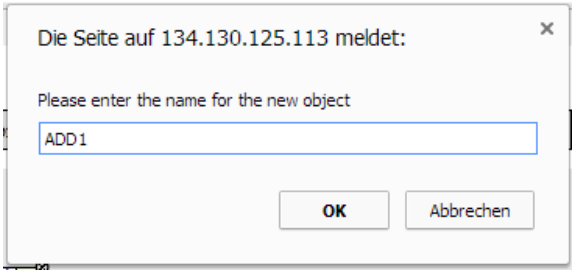

Abbildung 3: Aufforderung zur Eingabe des Namens der neu angelegten Instanz

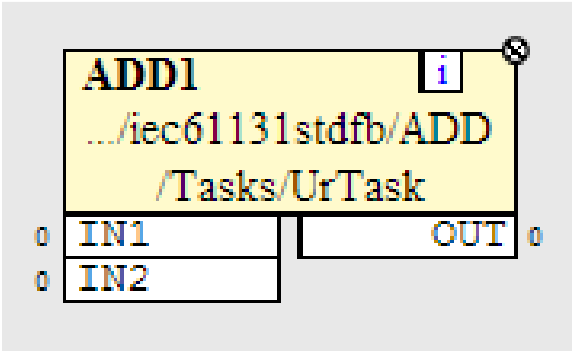

Abbildung 4: Neu angelegter Funktionsbaustein, nicht aktiviert

Durch Doppelklick auf einen Eingang öffnet sich ein Eingabefenster, in dem der Wert gesetzt werden kann. Auf diese Art können die Bausteine parametriert werden.

Durch einen Klick auf Con in der oberen Leiste (Abbildung 1) wir eine Verbindung angelegt. Diese ist zunächst nicht konfiguriert (Abbildung 5). Durch Rechtsklick auf die gewünschte Quellvariable wird diese eingerichtet (Abbildung 6). Durch einen weiteren Rechtsklick wird nun die Zielvariable eingerichtet. Dabei verschwindet das Konfigurationsfenster und die Verbindung wird als Linie angezeigt. Sie ist zunächst inaktiv (Abbildung 7), das heißt, sie überträgt keine Daten. Durch Rechtsklick auf die Verbindung wird sie aktiviert (Abbildung 8).

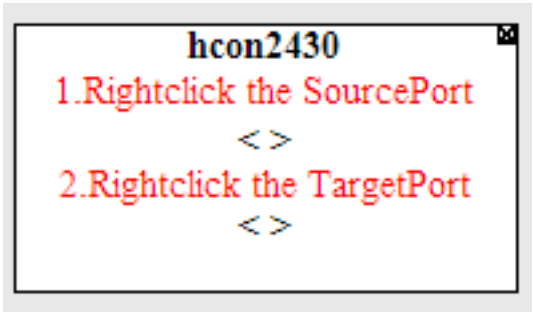

Abbildung 5: Nicht konfigurierte Verbindung

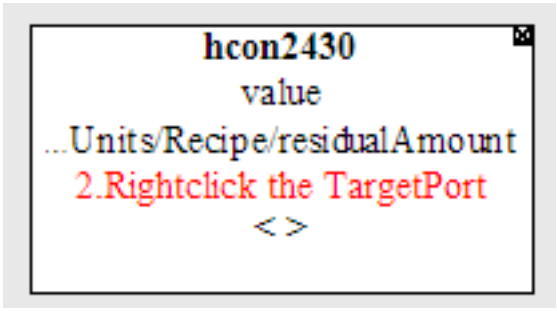

Abbildung 6: Verbindung mit konfigurierter Quelle

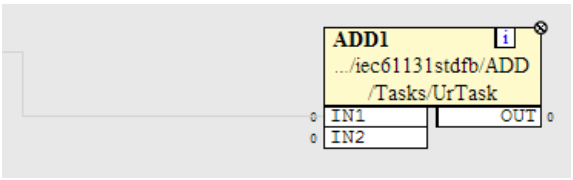

Abbildung 7: Vollständig konfigurierte Verbindung, inaktiv

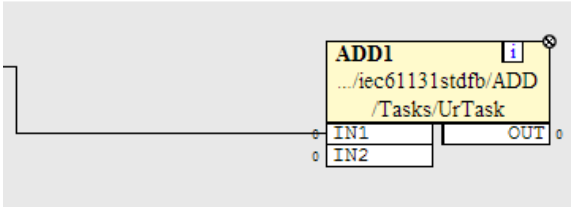

Abbildung 8: Vollständig konfigurierte Verbindung, aktiv

Durch einen Klick auf das i-Symbol oben im Funktionsbaustein (Abbildung 9) öffnet sich das in Abbildung 10 dargestellte Informationsfenster. Hier können weitere Einstellungen des Funktionsbausteins gemacht werden. Die Variable cyctime gibt an, in welchem Zyklus der Baustein ausgeführt werden soll. Der Zyklus kann immer nur ein Vielfaches des Zyklus des Elterntasks sein. Iexreq gibt an, ob dieser Baustein in jedem Zyklus (TRUE) oder nur bei Änderung eines Eingangswerts (FALSE) ausgeführt werden soll. Die Variable actimode sagt aus, ob der Baustein aktiviert ist. Dabei bedeutet 1 zyklisch ausführen, 2 einmalig ausführen und dann aus der Taskliste herausnehmen, 3 einmalig ausführen und dann deaktivieren und 0 nicht ausführen (deaktiviert). Die Aktivierung / Deaktivierung (Wechsel actimode 0 / 1) kann auch durch einen Rechtsklick auf den Kopf des Funktionsbausteins ausgelöst werden. Die Variablen Xpos und Ypos sind für das Darstellungssystem gedacht und speichern die Position an der der Funktionsbaustein angezeigt werden soll. Die Variable methcount ist ein Zähler der erfolgten Aufrufe dieses Funktionsbausteins.

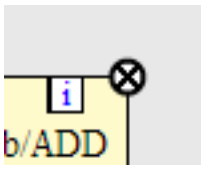

Abbildung 9: Info-Schaltfläche und Löschschaltfläche eines Funktionsbausteins

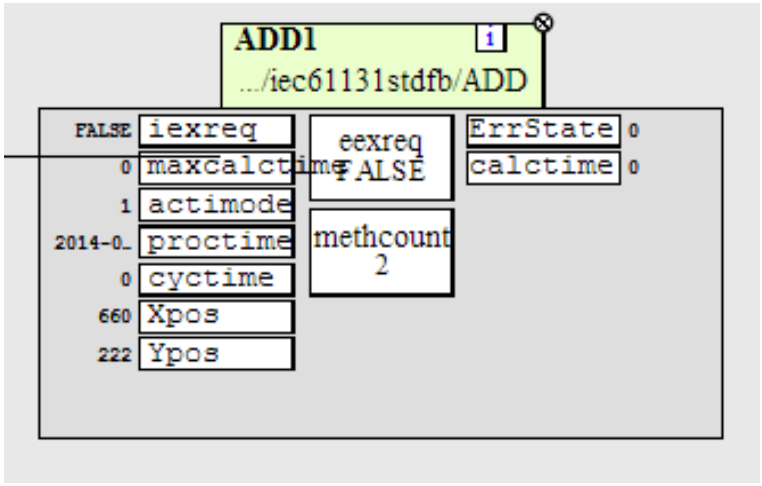

Abbildung 10: Informationsfeld eines Funktionsbausteins

Verbindungen können durch Doppelklick wieder gelöscht werden. Dabei wird eine Abfrage eingeblendet. Durch einen Klick auf das Kreuz in der oberen rechten Ecke eines Funktionsbausteins, wird dieser ebenfalls mit Nachfrage gelöscht.

2.3 Standardbibliotheken

Es gibt eine Reihe von Standardbibliotheken, die einen Großteil der wichtigen Funktionalitäten abdecken. Die Bibliothek *iec61131stdfb* stellt Funktionsbausteine für arithmetische und logische Operationen bereit. Auch Vergleiche sowie einfache Zähler und Zeitglieder sind vorhanden. Die Bibliothek *ACPLTlab003lindyn* stellt Bausteine mit linearer Dynamik (Integrator, PI-Glied, PT1-Glied, etc.) zur Verfügung. Die Bibliothek *IOdriverlib* stellt Abstraktionsbausteine für Feldanbindungen bereit. Des Weiteren gibt es Kommunikationsbausteine in der *fbcomlib*, Prozessführungsbausteine und Bausteine zum Senden und Empfangen einfacher Nachrichten in weiteren Bibliotheken.

2.4 Funktionsblätter

Die Bibliothek *fb* stellt neben der Basisklasse für Funktionsbausteine auch so genannte *functioncharts* zur Verfügung. Diese dienen zur Kapselung von Funktionsbausteinen. Sie geben die Möglichkeit, Funktionsbausteinnetze oder andere Modelle in ihrem containment anzulegen und stellen eine Taskliste bereit. Des Weiteren können sie zur Laufzeit um Eingangs- und Ausgangsports erweitert werden. Diese Ports verhalten sich wie die Ein- und Ausgänge von Funktionsbausteinen. Sie können also verbunden werden. Dadurch stellen Funktionsblätter die Möglichkeit bereit, Funktionsbausteinnetze und damit teils komplexe Funktionalität als einzelne Funktionsbausteine zu handhaben.

3 SSCs

Sequential State Charts dienen der Modellierung von Abläufen. Sie bilden eine Kette von Zuständen bzw. Schritten, zwischen denen bei der Erfüllung bestimmter Transitionsbedingungen gewechselt wird. Innerhalb der Schritte können Aktionen ausgeführt werden.

Als Ausgangspukt stellt die Bibliothek SSC dafür zwei Klassen: *controlchart* und *SequentialControlChart* bereit. Das *controlchart* ist von *functionchart* abgeleitet und erweitert dieses um ein Serviceinterface. *SequentialControlChart* erbt seinerseits von *controlchart* und ergänzt die

eigentliche SSC-Logik, d.h. die für das Verwalten und Ausführen der Schrittketten nötige Funktionalität.

Um einen SSC im RTE-System zu verwenden muss zunächst eine Instanz vom Typ *SequentialControlChart* erzeugt werden. Diese koordiniert die Abläufe und Zustandsübergänge. Anschließend werden Schritte und Transitionen, repräsentiert durch Intanzen der Klassen *Step* und *Transition*, erzeugt.

Bis hierher ist der Umgang mit dem Engineering-Werkzeug genau gleich wie bei Funktionsbausteinen im Allgemeinen. Dies ist der Tatsache geschuldet, dass das Engineering erstens nicht nur für Funktionsbausteine, sondern für alle Instanzen anwendbar ist. Für Funktionsbausteine kommen lediglich einige Automatismen hinzu, die dem Anwender die Arbeit erleichtern. Zweitens sind die für SSCs verwendeten Klassen größtenteils selbst Funktionsbausteine.

Sind die Schritte und Transitionen instanziiert, so müssen sie durch Anlegen der Links prevStep und nextStep bzw. prevTrans und nextTrans (je nachdem, von welcher Seite verlinkt wird) in die gewünschte Abfolge gebracht werden. Dabei kann aufgrund der verwendeten Links immer nur ein Schritt auf eine Transition folgen. Umgekehrt können aber mehrere Transitionen in denselben Schritt führen. Auch vor einer Transition darf nur ein Schritt sein, während mehrere Transitionen auf denselben Schritt folgen dürfen. Dies wird durch die Verwendung von 1-zu-n-Links erreicht, wobei das Elternobjekt immer ein Schritt ist. Auf diese Weise sind auch Alternativverzweigungen in einem SSC möglich. Parallelität ist in SSCs verboten. Dies wird dadurch wettgemacht, dass jeder Schritt mehrere Sub-SSCs ausführen darf.

Ist die gewünschte Abfolge festgelegt, so können die Transitionsbedingungen implementiert werden. Für diese stellt das *SequentialControlChart* eine Domain mit dem Namen .transConds bereit. Transitionsbedingungen sind Funktionsbausteine oder Funktionsblätter. Sie können für mehrere Transitionen verwendet werden. Transitionsbedingungen müssen einen booleschen Ausgang haben, der mit der Variable result der entsprechenden Transitionen verbunden wird.

Schritte können drei Typen von Aktionen ausführen: Variablen setzen, Kommandos (Prozessführungsserviceaufrufe) verschicken und Funktionsbausteine / Funktionsblätter ausführen. Die erste Möglichkeit wird durch einen Funktionsbaustein vom Typ *setVariable* implementiert. Dieser erhält als Parameter den Pfad der zu setzenden Variable (variable), den zu setzenden Wert (value) den actionQualifier, der angibt in welcher Phase eines Schrittes (einmalig bei Eintritt – 1; dauernd, solange der Schritt aktiv ist – 2; einmalig beim Verlassen – 3) die Aktion auszuführen ist, und einen actionName als Bezeichner der Aktion. Außer der Variable value sind alle Eingänge als Parameter ausgeführt, können also nicht mit anderen Bausteinen verbunden werden. Analog zum Setzen einer Variable können mit Bausteinen der Klasse *sendCommand* Prozessführungsserviceaufrufe verschickt werden. Über die Eingänge des Bausteins werden dafür das Ziel, der aufzurufende Service und etwaige Parameter festgelegt.

Die dritte Möglichkeit, das Ausführen von Funktionsbausteinen bzw. Blättern erfordert mehr Aufwand. Zunächst muss der auszuführende Baustein bzw. das auszuführende Blatt in der Domain .action im containment des *sscs* erzeugt werden. Dann wird ein Baustein vom Typ *executeFB* im containmenr des Schrittes angelegt, in dem die Aktion durchgeführt werden soll. Der *executeFB*-Baustein wird mit dem identifier des auszuführenden Bausteins / Blattes in der Variable actionName parametriert. Dabei wird automatisch ein Link vom Typ action zum benannten Baustein / Blatt

angelegt. Außerdem muss der actionQualifier gesetzt werden um die gewünschte Phase des Schrittes einzustellen.

Da Bedingungen und Aktionen innerhalb eines Anwendungsbereichs häufig ähnlich sind, ist es praktisch Teile von oder Ganze Bedingungen und Aktionen zu kopieren oder durch Parametrierung mehrfach verwendbar zu gestalten.

4 Aggregation von SSCs und Funktionsbausteinen

Im Folgenden wird zunächst der Aufbau eines kontinuierlichen Reglers als Führungselement aus SSC und Funktionsblättern dargestellt. Daran schließt sich die Beschreibung einer rezeptähnlichen Ablaufsteuerung als SSC an.

4.1 PI-Regler als gekapselte Führungsfunktionalität

Die Aufgabe des hier betrachteten Reglers ist die kontinuierliche Regelung des Sauerstoffgehalts einer Flüssigkeit über den zugeführten Gasstrom. Der Regler geht von einem Prozessführungsbaustein vom Typ SVAD (**S**ervo **V**alva with **A**nalog **D**rive) zur Aktoransteuerung aus. Der Ventilsteuerbaustein wird dienstbasiert angesteuert und der Sollwert über eine Netzwerkschnittstelle vorgegeben. Der Prozesszustand wird ebenfalls über eine Netzwerkschnittstelle abgefragt. Der Regler ist als Funktionsblatt umgesetzt, kann also wie ein Funktionsbaustein gehandhabt werden. Er besteht als Prozessführungseinheit aus zwei orthogonalen Automaten: dem Belegungszustandsautomat und der eigentlichen Funktionalität, die wiederum als Funktionsblätter umgesetzt sind. Abbildung 11 zeigt den Aufbau und die Verbindung des Reglers graphisch.

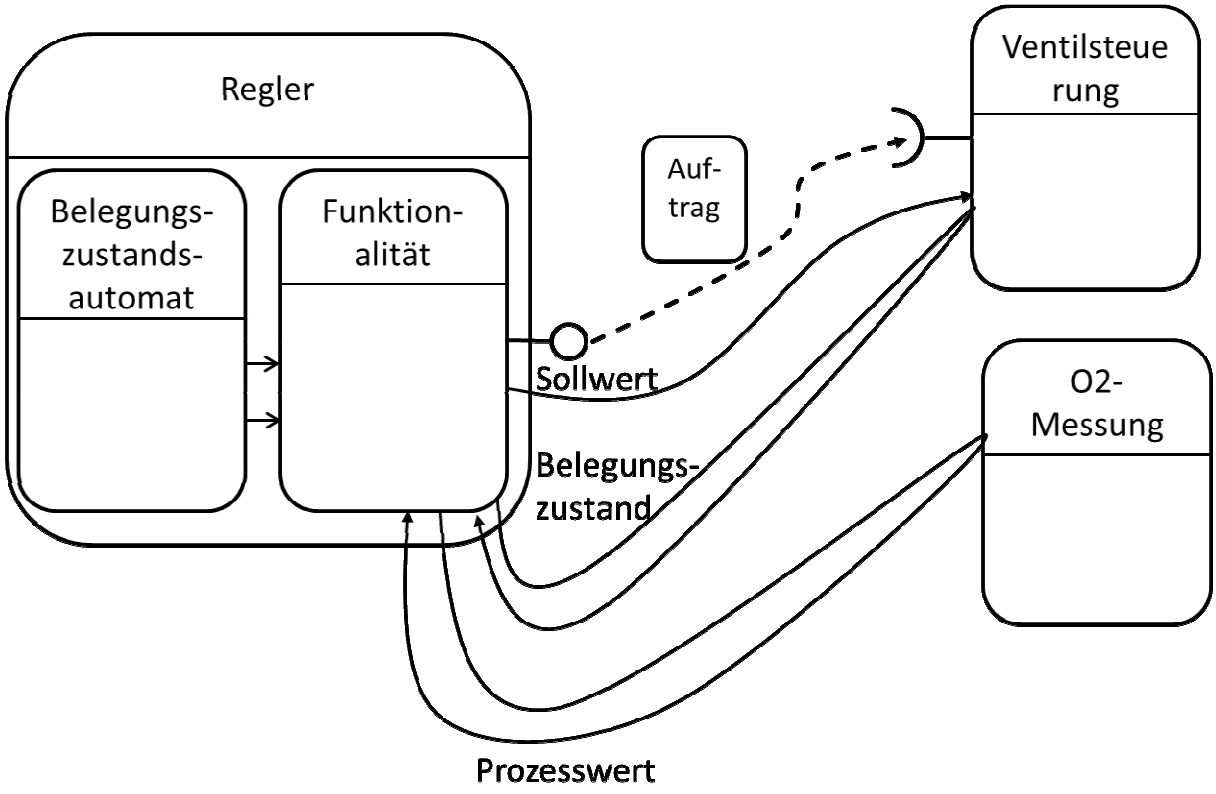

Abbildung 11: Aufbau und Einbindung des Reglers

Beide Automaten enthalten einen SSC. Da es sich beim Belegungszustandsautomat um eine Grundfunktion handelt, wird dieser so zu sagen von der Stange instanziiert. Es gibt hierfür ein Typical, das den Automaten gänzlich umsetzt. Der Belegungszustandsautomat kennt nur zwei Zustände: Frei und Belegt. Zwischen diesen wird je nach eingehendem Kommando gewechselt. Das Kommando „OCCUPY“ überführt den Zustandsautomaten vom Zustand frei in den Zustand belegt. Dabei wird der Kommandogeber als aktueller Beleger gesetzt. Das Kommando „FREE“ gibt die Einheit wieder frei. Ist eine Einheit belegt, so werden „OCCUPY“-Kommandos nicht akzeptiert, es sei denn, sie kommen vom Handbediener „OP“. Dieser darf die Einheiten jederzeit belegen. Gibt der Handbediener die Einheit wieder frei, so wechselt sie zum vorherigen Beleger zurück, sofern dieser gesetzt war. Die Überprüfung der Gültigkeit einer Kommandogeber Kommando Kombination erfolgt kontinuierlich im Funktionsbaustein usercheck. Dessen Ergebnis geht in die Transitionsbedingungen ein. Hier ist folglich SSC mit FBD kombiniert.

Die Zustände des Belegungszustandsautomaten werden in den Transitionen des Funktionalitätsautomaten ausgewertet. Dadurch kommt die Verbindung der beiden Automaten zu Stande.

Die Funktionalität als PI-Regler wird durch den zweiten SSC implementiert. Dieser hat die Zustände Off, Occupy, StartSCU, Control, StopSCU und Free. Der Automat setzt dabei folgende Funktionalität um:

- Zunächst befindet sich der Regler im Zustand Off
- Der Befehl SETPOINT aktiviert den Regler und gibt gleichzeitig einen neuen Sollwert vor.
- Wird der Regler aktiviert, so wechselt er in den Zustand Occupy und belegt die unterlagerte Ventilsteuereinheit.
- Ist die Ventilsteuereinheit belegt, so wird in den Zustand StartSCU gewechselt. In diesem Zustand wird die Ventilsteuereinheit aktiviert.
- Aus dem Zustand StartSCU wird sofort in den Zustand Control übergegangen.
- Bei Eintritt in den Zustand Control wird die Ventilsteuereinheit in den externen Modus geschaltet. Dann wird fortwährend der neue Sollwert von einem PI-Funktionsbaustein berechnet und per Netzwerkschnittstelle übertragen.
- Erhält der Regler im Zustand Control das Kommando STOP oder wird ein Interlock ausgelöst, so wechselt er in den Zustand StopSCU. In diesem Zustand wird der Stop-Befehl an die Ventilsteuerung gesendet.
- Aus dem Zustand StopSCU wird sofort in den Zustand Free übergegangen. In letzterem Zustand wird die Belegung der unterlagerten Ventilsteuerung aufgegeben.
- Aus dem Zustand Free wird dann sofort in den Zustand Off übergegangen.
- Erhält der Regler den Stop-befehl im Zustand Occupy, so geht er direkt in den Zustand Off über.

Abbildung 12 zeigt den Zustandsautomaten und die Funktionsblätter der Transitionsbedingungen und Aktionen.

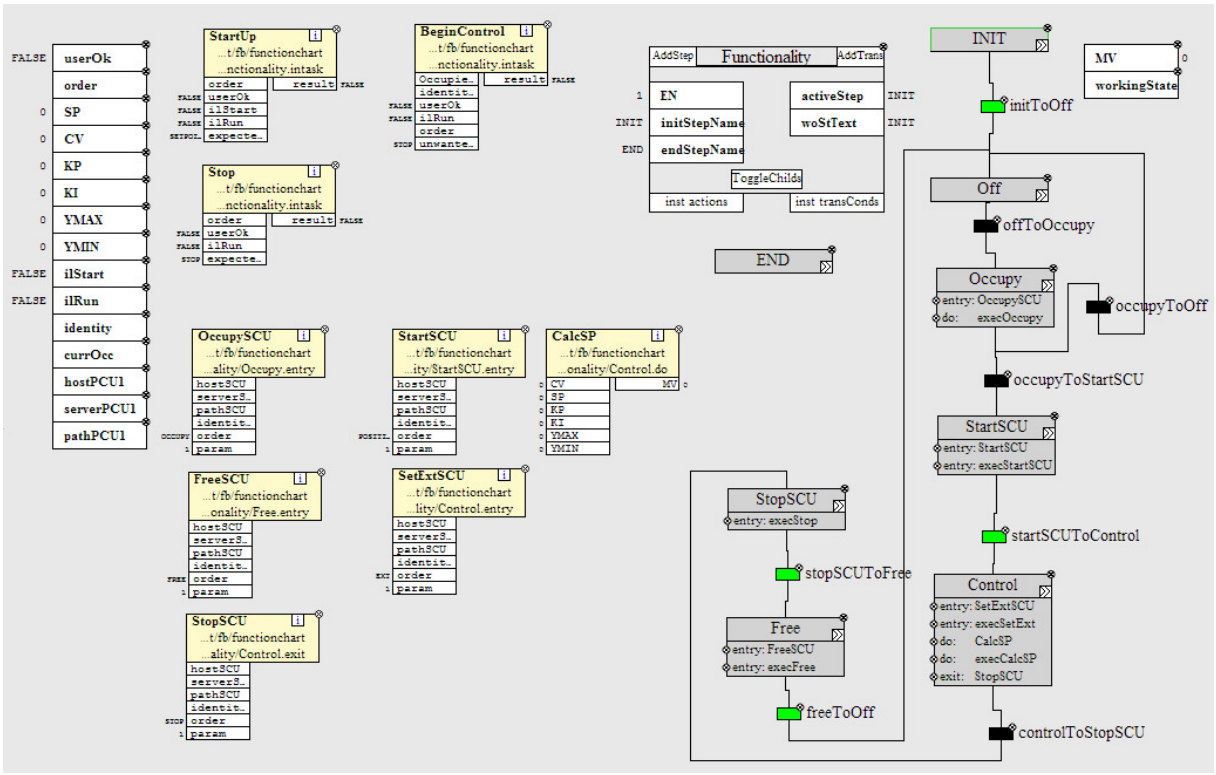

Abbildung 12: Aufbau der Funktionalität des Reglers. Rechts ist der Zustandsautomat mit Schritten und Transitionen abgebildet. Links oben sind die Funktionsblätter der Transitionsbedingungen (zu erkennen am Ausgang "result") und links unten die Funktionsblätter der Aktionen abgebildet.

4.2 Abläufe als gekapselte Führungsfunktionalität

Abläufe können in ähnlicher Weise wie der PI-Regler als gekapselte Einheiten umgesetzt werden. Es bedarf hier mindestens zweier Automaten: Belegungszustandsautomat und Funktionalitätsautomat. Der Belegungszustandsautomat kann aus dem vorigen Beispiel übernommen werden. Die Funktionalität hat im Falle des Ablaufs die Besonderheit, dass meist mehrere unterlagerte Einheiten angesteuert werden. Zusätzlich zu diesen Automaten kann eine Reihe weiterer Automaten verwendet werden, um den eigentlichen Ablauf zu parametrieren.

Zum Setzen von Parametern können zusätzliche Automaten eingesetzt werden. Sie reagieren auf ein parametrierbares Kommando und setzen eine Variable im SSC. Abbildung 13 zeigt beispielhaft den SSC zum setzen des PH-Sollwerts. Im Zustand Idle wartet der SSC auf das entsprechende Kommando. Trifft das Kommando ein, so feuert die Transitionsbedingung reactOnOrder und der SSC geht in den Zustand setValue über. In diesem Zustand wird die Parametervariable gesetzt und der SSC geht wieder in den Zustand Idle über.

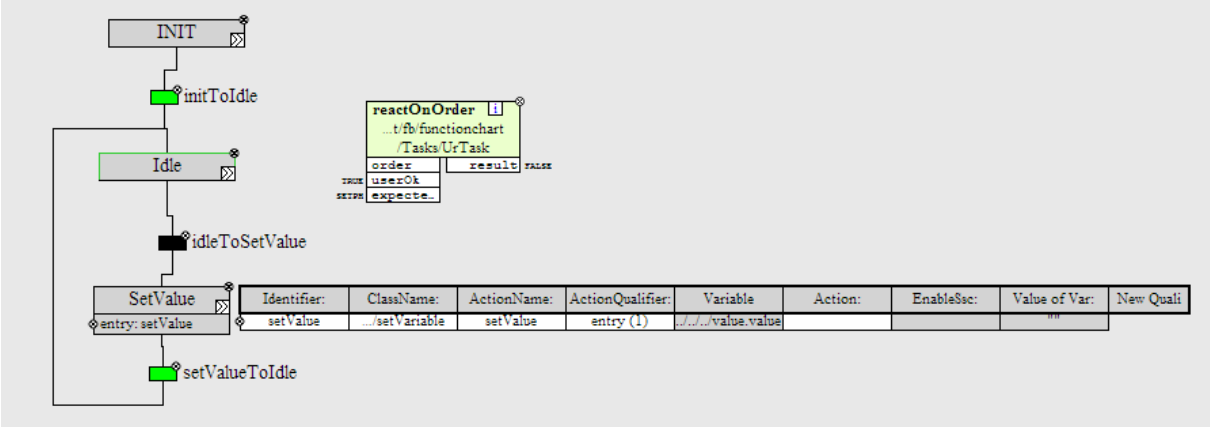

Abbildung 13: SSC zum Setzen eines Parameterwerts. Die Transitionsbedingung reactOnOrder reagiert auf ein parametrierbares Kommando und bewirkt den Übergang in den Zustand SetValue. Dieser setzt eine Ausgangvariable auf den Wert der Kommandonachricht. Danach geht der Automat sofort wieder in den Zustand Idle über. Die Beschreibung der setValue-Aktion ist neben dem Schritt dargestellt.

Im Falle eines Rezeptablaufes müssen verschiedene Einheiten angesteuert werden. Dazu bedient sich die Ablaufsteuerung der Dienstschnittstelle zur Auftragsvergabe an unterlagerte Einheiten und einer Netzwerkschnittstelle zur Abfrage von deren Belegungszuständen und Prozesswerten. Abbildung 14 zeigt diese Verbindungen grafisch.

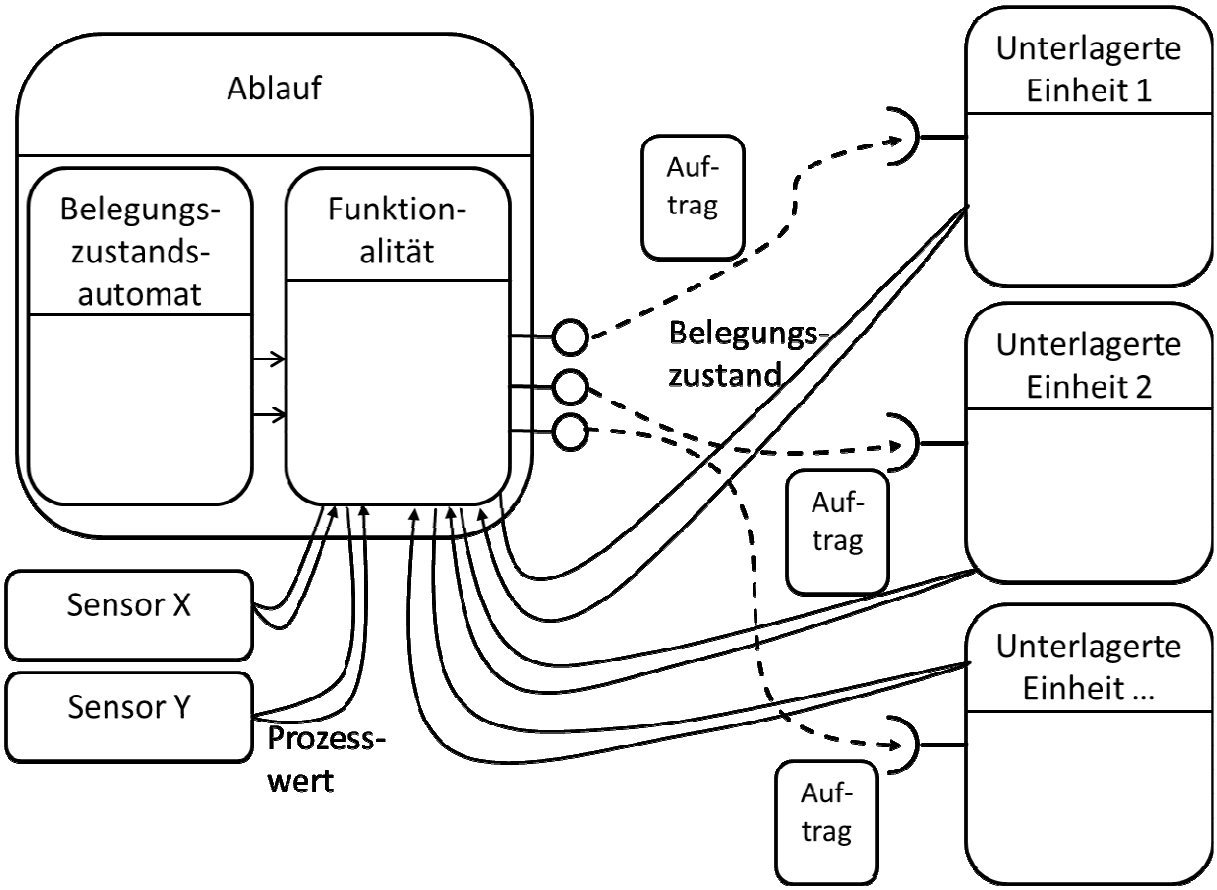

Abbildung 14: Verkoppelung von Ablaufsteuerung, unterlagerten Einheiten und Prozesssensorik

Auf die genaue Wiedergabe der zustände der Ablaufsteuerung soll hier verzichtet werden, da dies letztlich eine Wiederholung der Ablaufbeschreibung darstellt. Wichtiger ist anzumerken, dass derartige Ablaufsteuerungen nicht auf die Steuerung von Anlagen beschränkt sind. Da über definierte Schnittstellen (Dienste und ks) interagiert wird ist auch der Zugriff auf andere Modelle möglich.

Beispielsweise können zur Laufzeit die Zustände von Ventilen abgefragt werden und durch Verbindung mit einem PandiX-Modell Flusswege ermittelt werden. Auch die Manipulation von Modellen ist durch SSCs möglich.

Abbildungsverzeichnis