Kommunikation zwischen Funktionsbausteinen und UML Capsules in einer industriellen Softwareumgebung

Torsten Heverhagen, Jingyi Shi, Max von Groll, Rudolf Tracht
Universität Essen, FB 12, Automatisierungstechnik
Schützenbahn 70, 45127 Essen
{Torsten.Heverhagen | Max.von.Groll | Rudolf.Tracht}@uni-essen.de
shijingyi@automat.uni-essen.de

Themenschwerpunkt: C. Anwendungsbeispiele, Best Practice

Zusammenfassung: Mit Hilfe von Funktionsbausteinadaptern (FBAs) kann die Kommunikation zwischen Funktionsbausteinen der IEC 61131-3 und Capsules der Unified Modeling Language (UML) in einer formalen Sprache spezifiziert werden. Nachrichten, die von Capsules an Funktionsbausteine geschickt werden sollen, werden dabei in zeitabhängige Belegungen von Eingangsvariablen der Funktionsbausteine umgesetzt. Belegungen der Ausgangsvariablen von Funktionsbausteinen werden in Nachrichten umgewandelt, die an Capsules gesendet werden können. In diesem Artikel wird gezeigt, wie nach der Spezifikation ein solcher FBA mit Hilfe kommerziell verfügbarer Software-Werkzeuge implementiert wird.

1 Einleitung

Automatisierungstechnische Systeme müssen unter dem Druck eines stark veränderlichen Marktes immer flexibler gestaltet werden. Dazu bedarf es neben einer flexiblen Gerätetechnik besonders auch einer offenen und erweiterungsfähigen Softwarestruktur.

In weiten Bereichen der Automatisierungstechnik wurden und werden auf der IEC 61131 [1] basierende Automatisierungsgeräte verwendet. Wegen ihrer hohen Zuverlässigkeit, guten Echtzeiteigenschaften, großen Verbreitung und einfachen Handhabung kann davon ausgegangen werden, dass diese Art von Automatisierungsgeräten auch in Zukunft eingesetzt wird.

Immer komplexer werdende Anforderungen an die Steuerungssoftware erfordern jedoch in zunehmendem Maße den Einsatz objektorientierter Entwurfs- und Programmierumgebungen. Besonders in großen bestehenden Anlagen sind aber die Investitionskosten so hoch, dass bestehende Automatisierungsgeräte nicht einfach durch leistungsfähigere Industrie-PCs ersetzt werden können, was aber meist beim Einsatz objektorientierter Technologien erforderlich wäre.

Unser Ansatz ist es deshalb, objektorientierte Technologien in Teilsystemen großer Anlagen einzusetzen, die gerade erweitert oder modernisiert werden. Während unserer Arbeit an einer fertigungstechnischen Fallstudie, die in unserem Institut aufgebaut wurde, stellte sich heraus, dass es für die Modellierung von Kommunikation zwischen der von uns eingesetzten Modellierungssprache UML [2] und Sprachen der IEC 61131-3 noch wenig bis keine Unterstützung gab. Es zeigte sich, dass besonders in frühen Stadien des Systementwurfes eine möglichst hardware-unabhängige Beschreibung der Kommunikationsbeziehungen

zwischen beiden Systemen notwendig ist, da die gerätetechnische Ausführung der Schnittstellen später sehr unterschiedlich ausfallen kann.
Aus diesem Grund entwickelten wir einen neuen Stereotyp in der UML – den Function Block Adapter (FBA) [5]. Mit FBAs werden Funktionsbausteine so ummantelt, dass sie aus Sicht der UML wie Capsules (ein UML Stereotyp für aktive Objekte [2]) angesprochen werden können. Aus der Sicht der IEC 61131-3 kann ein FBA wie ein Funktionsblock behandelt werden. Ein FBA selbst beschreibt nur die logische Abbildung der unterschiedlichen Signal-/Nachrichten-Arten der beiden Sprachen aufeinander. Gerätetechnische Entscheidungen für die Verbindung zum Beispiel eines Industrie-PCs mit einem anderen Automatisierungsgerät werden erst während der Implementierung getroffen [6].
In diesem Beitrag soll über unsere Erfahrungen beim Einsatz von FBAs in einer konkreten fertigungstechnischen Anlage berichtet werden. In dieser Anlage wird ein Transportsystem mit Hilfe einer SPS (S7-315-2DP) gesteuert. Dieses Transportsystem muss mit einer Fertigungszelle kommunizieren, die in den Sprachen UML/C++ entwickelt und implementiert wurde. Die Kommunikation findet über einen Feldbus (PROFIBUS-DP) statt, der mit Hilfe eines RS232-Gateways mit der Fertigungszelle verbunden ist. Wir zeigen hierbei, wie ausgehend von den Anforderungen der Entwurf mit Hilfe von FBAs durchgeführt und anschließend mit Hilfe kommerziell verfügbarer Software-Werkzeuge (SIMATIC Step 7™ [4] und Rational Rose RealTime™ [3]) implementiert wird.

2 Anwendungsbeispiel: Fertigungsanlage

Das Anlagenschema der Fertigungsanlage ist in Abbildung 1 dargestellt.

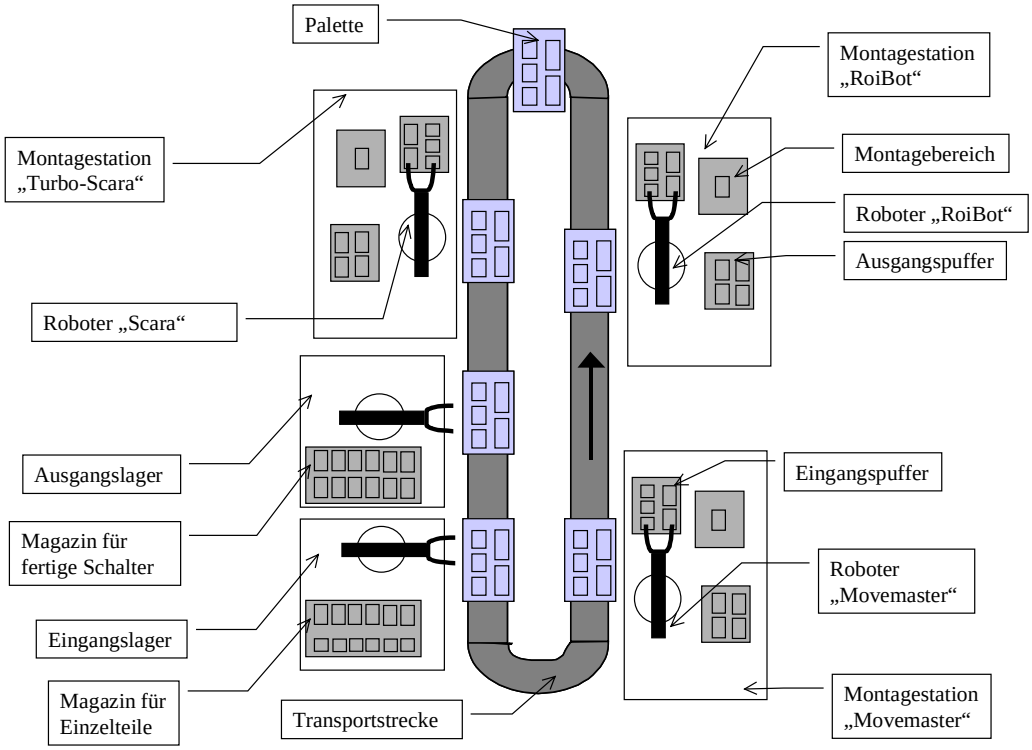

Abbildung 1. Schema der Fertigungsanlage

Drei Montagestationen werden über ein Transportsystem vom Eingangslager mit Einzelteilen beliefert. Die fertig montierten Produkte werden über das Transportsystem zum Ausgangslager gebracht. Dabei arbeiten die einzelnen Zellen (Montagestationen, Eingangs-, Ausgangslager und Transportsystem) weitestgehend autonom. Benötigt zum Beispiel eine Montagestation Einzelteile zur Montage eines Produktes, fordert sie diese beim Transportsystem an. Das Transportsystem ist dafür verantwortlich, eine geeignete Palette zum Eingangslager zu bewegen und diesem die Positionen innerhalb der Palette mitzuteilen, auf die Einzelteile gestellt werden sollen. Nachdem das Eingangslager seine Arbeit beendet hat, wird die Palette zur entsprechenden Montagestation gefahren. Der Montagestation wird vom Transportsystem die Ankunft einer Palette mitgeteilt. Für den Informationsaustausch zwischen Transportsystem und den Roboterzellen (Lager oder Montagestation) werden drei unterschiedliche Typen von Nachrichten verwendet:

- Die Transportanforderung (Transport Request, TRQ) wird von einer Montagestation an das Transportsystem gesendet. Die Nachricht TRQ beinhaltet Informationen über die Art und Anzahl der benötigten oder abzutransportierenden Einzelteile oder Produkte.
- Die Antwort auf eine Transportanforderung (Transport Response, TRS) wird vom Transportsystem an eine Roboterzelle gesendet, wenn Einzelteile oder Produkte vor einer Roboterzelle angekommen sind. Die Nachricht TRS beinhaltet Informationen über die Art der Teile und deren Positionen auf der Palette.
- Die Palettenfreigabe (Pallet Free, PF) wird von einer Roboterzelle an das Transportsystem gesendet, wenn alle Teile von der Palette entnommen bzw. auf die Palette gestellt wurden und sich der Roboterarm aus dem Kollisionsbereich der Palette entfernt hat. Als Information wird dieser Nachricht nur die Nummer der Roboterzelle mitgegeben, damit das Transportsystem die Nachricht einer Roboterzelle zuordnen kann.

Mit diesen drei Nachrichten haben wir die Kommunikation zwischen Transportsystem und Roboterzellen zunächst nur informell beschrieben. Im weiteren Verlauf dieses Beitrages möchten wir eine formale Beschreibung vorstellen. Dazu stellen wir in Abschnitt 2.1 die Softwareschnittstelle des Transportsystems in Form eines Funktionsbausteines dar. In Abschnitt 2.2 erläutern wir dann die Softwareschnittstelle einer Roboterzelle, die in Form einer UML Capsule-Klasse vorliegt.

2.1 Software-Schnittstelle des Transportsystems

Die Steuerung des Transportsystems ist in Form eines Funktionsbausteines (FB) gegeben, der in Abbildung 2 zu sehen ist. Der FB besitzt drei Gruppen von Eingangs- und Ausgangsvariablen, deren Namen jeweils mit dem Prefix *TRQ*, *Pallet_Free* oder *TRS* beginnen. Die Variablen *TRQ_In*, *TRQ_Out* und *TRQ_Data_In*

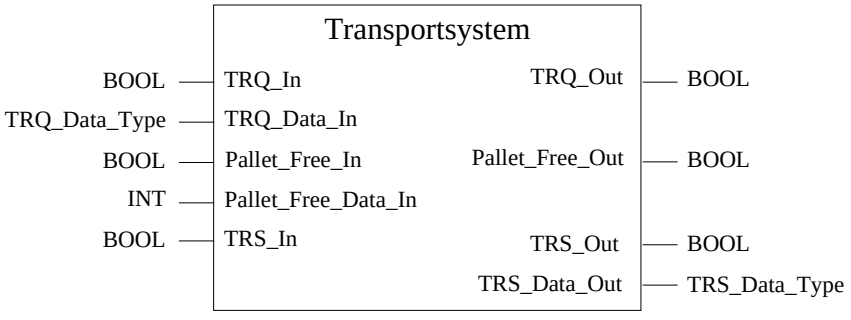

Abbildung 2. Funktionsbaustein Transportsystem

werden vom Transportsystem zur Entgegennahme der Transportanforderung verwendet. Die Variablen *Pallet_Free_In*, *Pallet_Free_Out* und *Pallet_Free_Data_In* werden für die Nachricht Palettenfreigabe benutzt. Mit den Variablen *TRS_In*, *TRS_Out* und *TRS_Data_Out* kann das Transportsystem eine Antwort auf eine Transportanforderung versenden. *TRQ_In* und *Pallet_Free_In* zeigen dem Transportsystem den Beginn einer Nachrichtenübermittlung von einer Roboterzelle an. Mit *TRS_Out* startet das Transportsystem eine Nachrichtenübermittlung an eine Roboterzelle.
Um das Beispiel aus den Abschnitten 3 und 4 zu unterstützen, soll hier die Antwort auf eine Transportanforderung (TRS) genauer erläutert werden. Dazu dient das Zeitdiagramm aus Abbildung 3. Die Nachricht TRS besitzt die höchste Priorität. Da der Kommunikationskanal zwischen Transportsystem und Roboterzelle nicht die gleichzeitige Übermittlung mehrerer Nachrichten erlaubt, muss die Übertragung von Nachrichten mit niedrigerer Priorität abgebrochen werden, falls diese zur gleichen Zeit gesendet wurden.

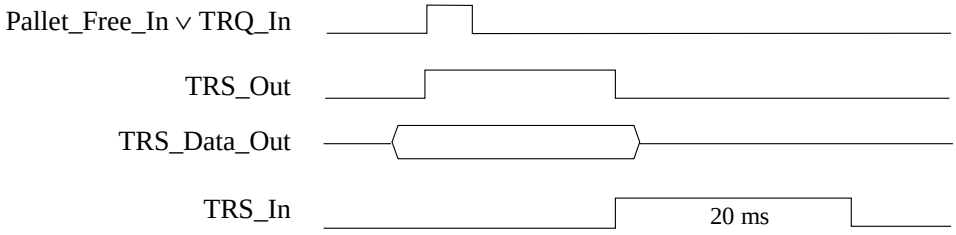

Abbildung 3. Zeitdiagramm zur Nachricht TRS

Die Daten, die mit TRS gesendet werden sollen, werden vom Transportsystem in *TRS_Data_Out* zur Verfügung gestellt. Mit einer positiven Flanke in *TRS_Out* wird dem Empfänger signalisiert, dass die Daten zur Verfügung stehen. Der Empfänger bestätigt dann in *TRS_In* die Übernahme der Daten durch eine positive Flanke. *TRS_In* soll frühestens nach 20ms wieder zurückgesetzt werden.

TRS_Data_Out ist vom Typ *TRS_data_type*. Das ist ein benutzerdefinierter Datentyp nach IEC 61131-3 (Abbildung 4). Er beinhaltet die ID der Transportanforderung, zu der die Antwort gehört (*TRQ_ID*), die ID der Roboterzelle, an die die TRS gesendet werden soll (*TRS_Robot*) und die Informationen über die Beladung der Palette (*TRS_pos*).
Jede Palette besitzt 7 Positionen, auf denen Teile platziert werden können. *TRS_pos* ist ein Feld, das für jede Position einer Palette eine Datenstruktur vom Typ Pos_info enthält. Das Transportsystem kann eine Roboterzelle auffordern, ein Teil auf eine Position einer Palette zu

```
(1)  TYPE  TRS_data_type
(2)  STRUCT
(3)    TRQ_ID: INT := 0;
(4)    TRS_Robot: INT := 0;
(5)    TRS_pos: array[1..7] of Pos_info;
(6)  END_STRUCT
(7)  END_TYPE

(8)  TYPE Pos_info
(9)  STRUCT
(10)   Pos_Status: BOOL := FALSE;
(11)   Part_on: Part_info;
(12) END_STRUCT
(13) END_TYPE

(14) TYPE Part_info
(15) STRUCT
(16)   Part_Type: INT := 0;
(17)   Part_Sender: INT := 0;
(18)   Part_Receiver: INT := 0;
(19) END_STRUCT
(20) END_TYPE
```

Abbildung 4. Definition der Datentypen zu TRS

stellen, oder eines zu entnehmen. Soll ein Teil auf die Position gestellt werden, ist *Pos_Status* gleich *FALSE*. In *Part_on* befinden sich genauere Informationen über die Art des Teiles, das auf der Position zur Verfügung steht bzw. auf diese gestellt werden soll. In *Part_Type* wird die Typnummer des Teiles gespeichert. *Part_Sender* und *Part_Receiver* werden zur Zeit nur intern des Transportsystems verarbeitet und deshalb nicht an eine Roboterzelle weitergeleitet (siehe Abschnitt 3).
Im nächsten Abschnitt wird der Kommunikationspartner dieses Funktionsbausteines und dessen Schnittstelle vorgestellt.

2.2 Software-Schnittstelle einer Roboterzelle

Die Steuerung einer Roboterzelle ist in Form eines UML-Capsules gegeben. Ein Capsule ist einer UML-Klasse ähnlich. Man verwendet es zur Modellierung der Struktur und des Verhaltens von Objekten. Im Unterschied zu normalen Klassen dürfen Capsules aber keine öffentlichen (public) Attribute oder Operationen besitzen. Eine Kommunikation zwischen Capsules ist nur über sogenannte Ports möglich. Um Nachrichten über solche Ports versenden und empfangen zu können, müssen diese Nachrichten in Protokollen definiert werden. Protokolle sind ebenfalls eine spezielle Art von Klassen.

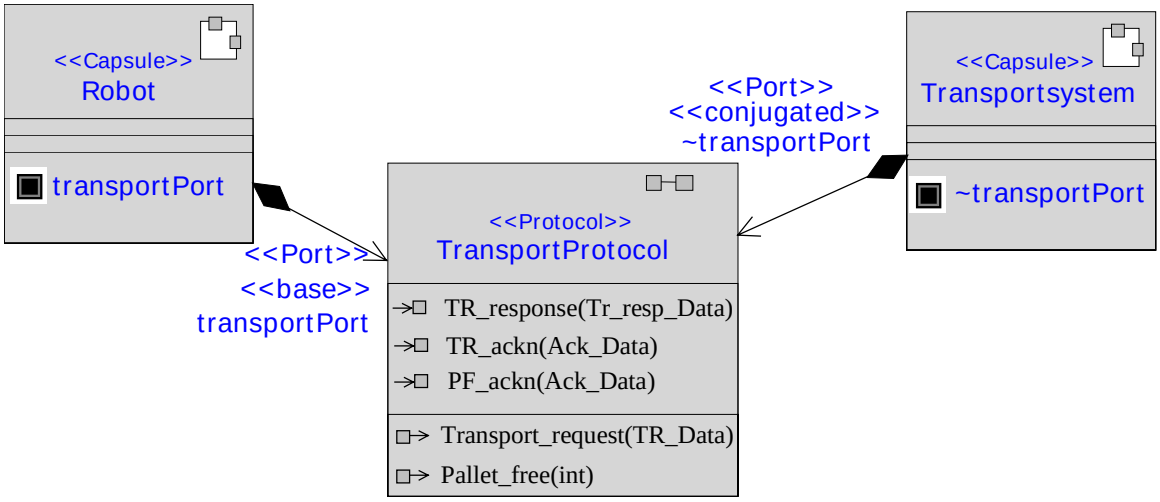

Abbildung 5. Klassendiagramm zum Transportprotokoll

Abbildung 5 zeigt ein UML-Klassendiagramm, das das Protokoll *TransportProtocol* und zwei Capsules *Robot* und *Transportsystem* enthält. *Robot* besitzt ein Port *transportPort*, das dem *TransportProtocol* zugeordnet ist. Deshalb kann es die Nachrichten *TR_response*, *TR_ackn*, *PF_ackn* empfangen und die Nachrichten *Transport_request* bzw. *Pallet_free* senden. Das Transportsystem soll die Nachrichten *Transport_request* und *Pallet_free* empfangen und die restlichen Nachrichten senden können. Deshalb muss es einen Port (*~transportPort*) besitzen, das die umgekehrte (*conjugated*) Rolle des *TransportProtocol* implementiert. Damit können die beiden Capsules miteinander kommunizieren. Eine ausführlichere Beschreibung der Konzepte Capsule, Port und Protocol findet sich in [2].
Das Transportsystem liegt natürlich wie in Abschnitt 2.1 vorgestellt als Funktionsbaustein vor. Wir haben es in Abbildung 5 als Capsule eingeführt, um das Konzept des „conjugated“ Port zu erläutern. Im Abschnitt 3 ersetzen wir das Capsule *Transportsystem* mit einem

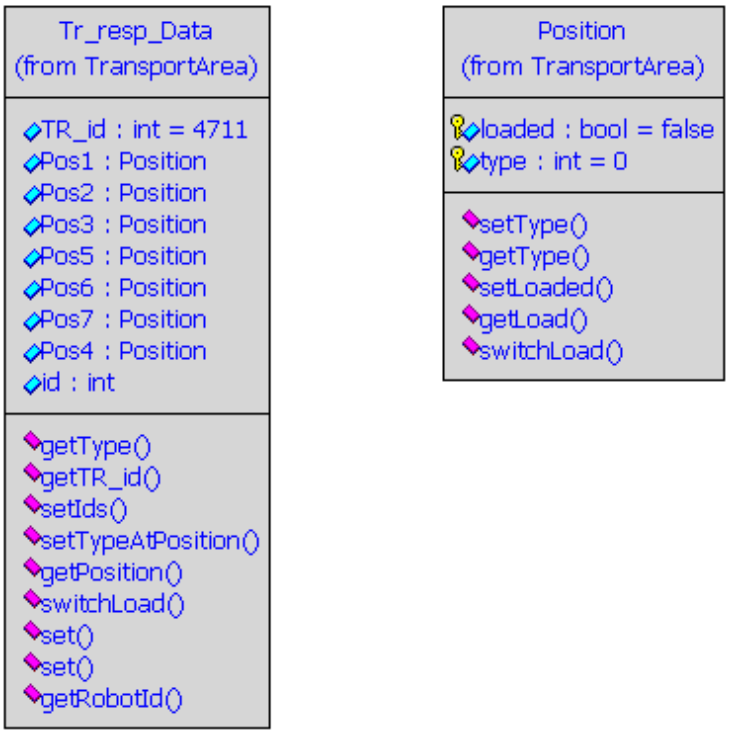

Abbildung 6. Klassendiagramm zu Datenklassen

Funktionsbausteinadapter. Vorher sollen aber noch die Daten, die mit den Nachrichten versendet werden, näher erläutert werden.

Jede Nachricht eines Protokolls kann optional Daten beinhalten. Die Antwort auf eine Transportanforderung beinhaltet zum Beispiel Informationen über Teile, die auf eine Palette gestellt oder von einer Palette entnommen werden sollen und deren Positionen (siehe vorheriger Abschnitt). Im *TransportProtocol* heißt diese Nachricht *TR_response*. Mit dieser Nachricht kann man eine Instanz der Klasse *Tr_resp_Data* verschicken. Diese Klasse ist in Abbildung 6 in Form eines UML-Klassendiagramms abgebildet.

Ebenso wie der strukturierte Datentyp *TRS_data_type* aus Abschnitt 2.1 beinhaltet *Tr_resp_Data* die ID des Roboters (*id*) und die ID der Transportanforderung (*TR_id*). Für jede Position auf der Palette besitzt die Klasse ein Attribut vom Typ *Position*. *Position* ist eine Klasse, die in Abbildung 6 rechts dargestellt ist. Sie enthält zwei Attribute *loaded* und *type*. *loaded* ist *false*, wenn die Position leer ist und *true*, wenn ein Teil auf der Position der Palette steht. Ist *type* gleich Null, dann soll die Roboterzelle diese Position nicht bedienen. Ist in *type* die ID eines Teiletyps gespeichert, soll ein Teil dieses Typs auf diese Position gestellt oder von dieser Position entnommen werden (entsprechend *loaded*). Da die beiden Attribute der Klasse *Position* nicht öffentlich sind, kann man auf sie nur über Operationen wie *setType*, *getType*, *setLoaded* und *getLoad* zugreifen. Obwohl alle Attribute der Klasse *Tr_resp_Data* öffentlich sind, wurden in dieser Klasse ebenfalls Zugriffsoperationen zur Verfügung gestellt. Mit der Operation *setIDs* kann man zum Beispiel die beiden IDs für die Roboterzelle und die Transportanforderung setzen (siehe Abbildung 9).

Eine Roboterzelle erwartet eine Antwort auf eine Transportanforderung als UML-Nachricht der Form, wie wir sie in diesem Abschnitt beschrieben haben. Da die Steuerung des Transportsystems aber als Funktionsbaustein vorliegt, muss die FB-Nachircht *TRS* geeignet in die UML-Nachricht *Tr_response* umgesetzt werden. Das Gleiche gilt auch für die anderen Nachrichten, die im Transportprotokoll enthalten sind. Die formale Spezifikation dieser Umsetzung wird im folgenden Abschnitt vorgestellt. Abschnitt 4 beschreibt die Implementierung dieser Kommunikationsbeziehungen.

3 Modellierung der Kommunikationsbeziehungen

Um ein Capsule mit einem Funktionsbaustein verbinden zu können, haben wir einen Funktionsbausteinadapter (FBA) eingeführt (Abbildung 7). Die Attribute des Adapters sind die Eingangs- und Ausgangsvariablen des Funktionsbausteines (FB), wobei die Schreibweise der bei IEC 61131-3 üblichen entspricht. Ausgangsvariable des FB sind Eingangsvariable des FBA und umgekehrt.

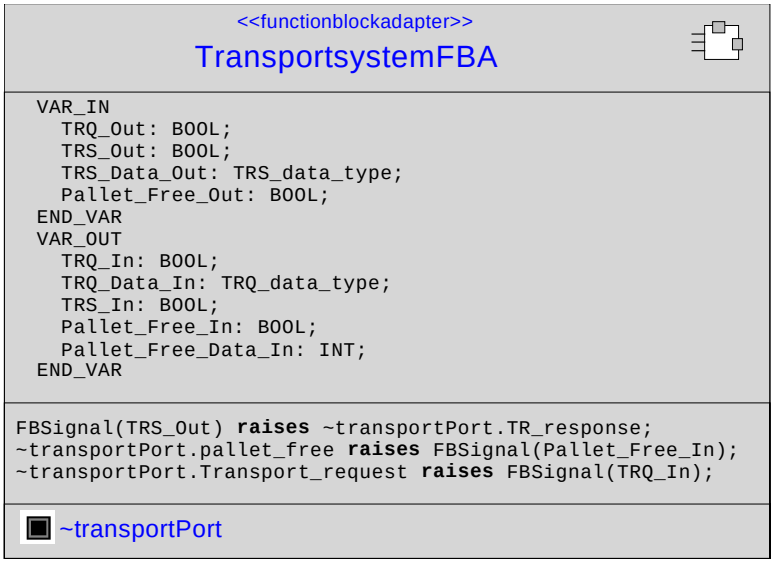

Abbildung 7. TransportsystemFBA

Das Port *~transportPort* des FBA implementiert die konjugierte Rolle des Protokolls *TransportProtocol*. Damit kann das Port mit dem des Roboter-Capsules verbunden werden. Genauso können die Eingangs- und Ausgangsvariablen des FBA mit denen des FB verbunden werden (Abbildung 8).

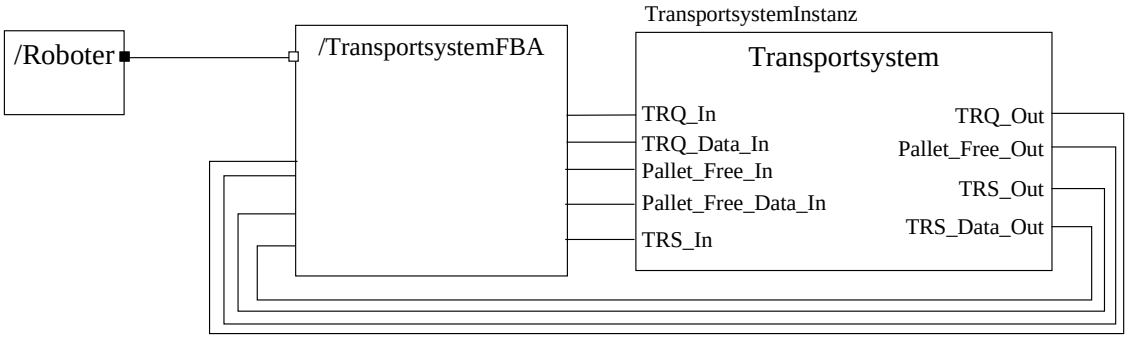

Abbildung 8. Strukturdiagramm mit Funktionsbaustein und FBA

Abbildung 8 zeigt eine Kombination aus UML-RT Strukturdiagramm und dem Funktionsblockdiagramm nach IEC 61131-3. Damit ist die statische Struktur des FBA vollständig festgelegt. Das dynamische Verhalten des FBA wird durch dessen Operationen bestimmt. In Abbildung 7 erkennt man im zweiten Listenbereich, in dem bei normalen UML-Klassen die Operationen stehen, drei Zeilen mit dem Schlüsselwort *raises*. Links von *raises* steht ein UML- bzw. FB- Signal, das in das FB- bzw. UML- Signal rechts von *raises* übersetzt wird. Mit dem Schlüsselwort *FBSignal* links von *raises* werden Boolesche Ausdrücke definiert, die der FBA zur Erkennung des Startes einer Nachrichtenübertragung vom Funktionsbaustein nutzt. Rechts von *raises* steht *FBSignal*, um festzulegen, welche Boolesche Bedingung den Start einer Nachrichtenübertragung an den Funktionsbaustein signalisiert. Die UML-Signale sind in der Schreibweise *Portname.Signalname* notiert.

Jeder *raises*-Anweisung wird eine FBA-Operation zugeordnet. In einer FBA-Operation wird die zeitliche Belegung der Eingangs- und Ausgangsvariablen des FBA beschrieben.

```
(1)    On_FBSignal(TRS_Out)
(2)    Signals
(3)      s1: ~transportPort.TR_response;
(4)    Begin
(5)      s1.setIds( TRS_Data_Out.TRS_Robot, TRS_Data_Out.TRQ_ID);
(6)      s1.Pos1.setType( TRS_Data_Out.Position[1].Part_on.Part_type);
(7)      s1.Pos1.setLoaded( TRS_Data_Out.Position[1].Pos_Status);
(8)      s1.Pos2.setType( TRS_Data_Out.Position[2].Part_on.Part_type);
(9)      s1.Pos2.setLoaded( TRS_Data_Out.Position[2].Pos_Status);
(10)     s1.Pos3.setType( TRS_Data_Out.Position[3].Part_on.Part_type);
(11)     s1.Pos3.setLoaded( TRS_Data_Out.Position[3].Pos_Status);
(12)     s1.Pos4.setType( TRS_Data_Out.Position[4].Part_on.Part_type);
(13)     s1.Pos4.setLoaded( TRS_Data_Out.Position[4].Pos_Status);
(14)     s1.Pos5.setType( TRS_Data_Out.Position[5].Part_on.Part_type);
(15)     s1.Pos5.setLoaded( TRS_Data_Out.Position[5].Pos_Status);
(16)     s1.Pos6.setType( TRS_Data_Out.Position[6].Part_on.Part_type);
(17)     s1.Pos6.setLoaded( TRS_Data_Out.Position[6].Pos_Status);
(18)     s1.Pos7.setType( TRS_Data_Out.Position[7].Part_on.Part_type);
(19)     s1.Pos7.setLoaded( TRS_Data_Out.Position[7].Pos_Status);
(20)     sendAsync(s1);
(21)     TRS_In := true;
(22)     delay(T#20ms);
(23)     TRS_In := false;
(24)   End_On_FBSignal
```

Abbildung 9. Operation *On_FBSignal(TRS_Out)*

Abbildung 9 zeigt beispielhaft, wie die Umsetzung der Antwort auf eine Transportanforderung vom FB-Signal in ein UML-Signal durch eine FBA-Operation beschrieben wird. Als erstes definiert die Operation eine Instanz *s1* des Signals *TR_response*. Dann werden die Daten von *s1* entsprechend der Daten von *TRS_Data_Out* gesetzt. Dazu kann *s1* wie eine Instanz der Klasse *TR_resp_Data* betrachtet werden. Der Zugriff auf Elemente der strukturierten Variable *TRS_Data_Out* erfolgt wie in der nach IEC 61131-3 üblichen Schreibweise. Es dürfen dabei nur Variablen elementaren Datentyps einander zugewiesen werden, weil für diese Datentypen eine Abbildung zwischen UML und IEC 61131-3 innerhalb des FBA-Frameworks existiert. Nachdem alle Daten umgesetzt wurden, kann das Signal mit *sendAsync* gesendet werden. Zum Schluss wird dem Funktionsbaustein mit einer positiven Flanke in *TRS_In* die erfolgreiche Nachrichtenübermittlung bestätigt.

Der FBA enthält noch zwei weitere Operationen für die UML-Signale *Transport_request* und *pallet_free*. Diese Operationen besitzen einen ähnlichen Aufbau mit dem Unterschied, dass *TRQ_Data_In* und *Pallet_Free_Data_In* Werte aus den UML-Signalen zugewiesen werden müssen.

Damit ist das Verhalten und die Struktur des FBA auf logischer, hardwareunabhängiger Ebene vollständig beschrieben. Diese Beschreibung kann nun als formale Spezifikation für die weitere Implementierung der Schnittstelle zwischen Capsule und Funktionsbaustein verwendet werden. Im nächsten Kapitel soll ein Eindruck vermittelt werden, wie die Implementierung durch den Einsatz kommerziell verfügbarer Software-Werkzeuge durchgeführt werden kann.

4 Implementierung der Kommunikationsbeziehungen

Der interne Aufbau des TransportsystemFBA muss der Verteilung des FBA über zwei Computersysteme Rechnung tragen. Dieser Sachverhalt ist in Abbildung 10 dargestellt.

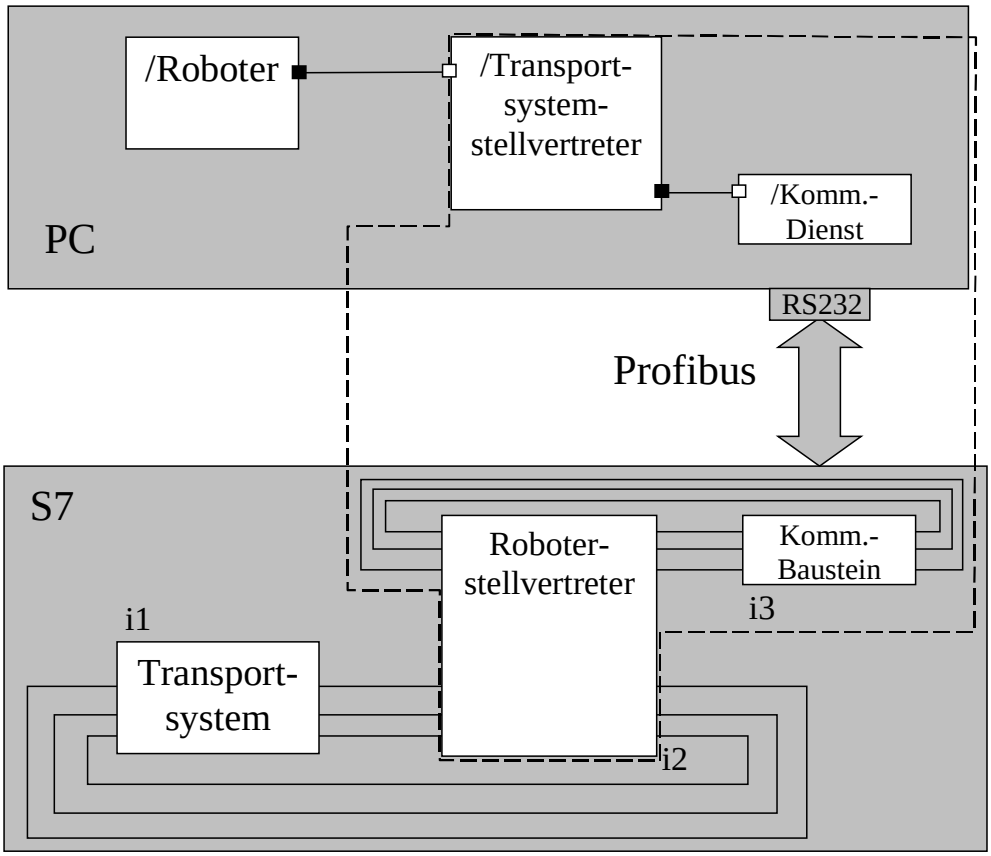

Abbildung 10. Struktur für die Implementierung des FBA

Wie wir in der Einleitung bereits erwähnt haben, ist die Steuerung des Transportsystems auf einer SPS als Funktionsbaustein implementiert. Die Steuerung der Roboterzelle hingegen befindet sich auf einem PC und liegt als Capsule-Klasse vor. Die Instanz des Funktionsbausteines Transportsystem ist deshalb in Abbildung 10 in der SPS (S7) dargestellt (Instanzname *i1*) während die Instanz der Capsule-Klasse *Roboter* auf dem PC ausgeführt wird. PC und SPS kommunizieren über ein Feldbussystem (PROFIBUS-DP) miteinander. Die Bestandteile des in Abschnitt 3 eingeführten FBAs sind in Abbildung 10 gestrichelt umrandet dargestellt. Es gibt zum einen ein Capsule, das innerhalb des FBA auf der Seite des PC als Stellvertreter für das Transportsystem fungiert, und zum anderen einen Funktionsbaustein, der auf der Seite der SPS als Stellvertreter für die Robotersteuerung arbeitet. Diese beiden „Stellvertreterbausteine“ verwenden plattformabhängige Kommunikationsdienste/-bausteine, um untereinander Nachrichten auszutauschen.

Der PC ist über eine serielle Schnittstelle mit einem RS232-Gateway des Profibus-DP verbunden. Zum Ansprechen dieser Schnittstelle werden die Windows-API Funktionen zur Dateibearbeitung benötigt. Um einen Polling-Betrieb erlauben zu können, muss die serielle Schnittstelle im asynchronen Modus betrieben werden. Das bedeutet, dass das Lesen und Schreiben über die Schnittstelle in einem Hintergrundprozess verläuft, während die

Anwendung (in Abbildung 10 ist das der *Transportsystemstellvertreter*) über die Windows-API Funktion *GetOverlappedResult* den Status der Lese- bzw. Schreiboperationen abfragen kann.
Die SPS (S7 315-2DP) ist sehr eng mit dem Profibus-DP gekoppelt. Die Eingangs- und Ausgangsdatenbereiche der Profibus-Slaves (z.B. des RS232-Gateways) werden automatisch auf den Eingangs- und Ausgangsdatenbereich der SPS abgebildet. Über Statusbits wird der SPS mitgeteilt, ob das RS232-Gateway gerade Daten sendet oder empfängt.
Für den Transportsystem- bzw. Roboterstellvertreter möchten wir im Folgenden C-Prozess (C für Capsule) und F-Prozess (F für Funktionsbaustein) schreiben. Das soll verdeutlichen, dass innerhalb des FBA zwei nebenläufige Prozesse synchronisiert werden müssen. Dazu ist weiterhin ein FBA-internes Kommunikationsprotokoll notwendig, das zwischen F-Prozess und C-Prozess über den Profibus betrieben werden muss.
Der C-Prozess hat die Aufgaben

- Kommunikation mit dem Roboter-Capsule über den *~transportPort* entsprechend dem *TransportProtocol*,
- Kommunikation mit dem F-Prozess über den Profibus entsprechend dem FBA-internen Protokolls und
- Konvertierung der Daten aus dem *TransportProtocol* in das FBA-interne Protokoll und umgekehrt.

Der F-Prozess hat die Aufgaben

- Kommunikation mit dem Funktionsbaustein Transportsystem über die Schnittstellenvariablen entsprechend des vereinbarten FB-Protokolls,
- Kommunikation mit dem C-Prozess über den Profibus entsprechend dem FBA-internen Protokolls und
- Konvertierung der Daten aus dem FB-Protokoll in das FBA-interne Protokoll und umgekehrt.

In der FBA-internen Kommunikation muss das Eintreffen einer externen Nachricht dem jeweils anderen Prozess mitgeteilt und zugehörige Daten übertragen werden. Weiterhin müssen Konfliktfälle behandelt werden, wenn zum Beispiel die Roboterzelle und das Transportsystem gleichzeitig eine Nachricht übermitteln möchten. Für solche Fälle müssen die Nachrichten mit Prioritäten versehen werden (siehe Abschnitt 2.1). In der Konfliktbehandlung wird dann die Übermittlung der Nachricht mit niedrigerer Priorität abgebrochen. Das wird anschließend dem Sender der niedrigprioren Nachricht mitgeteilt. Die Datenkonvertierung in das bzw. aus dem FBA-internen Protokoll ist im Wesentlichen eine Serialisierung bzw. Deserialisierung der Datenstrukturen des Funktionsbausteines und des Capsules. Dabei muss man zum Beispiel darauf achten, dass eine Variable vom Typ *INT* nach IEC 61131-3 nur zwei Byte lang ist und damit einer C++-Variable vom Typ *short int* entspricht. Außerdem ist die Reihenfolge von höher- und niederwertigem Byte in beiden Programmiersprachen unterschiedlich.
Das Verhalten des C-Prozesses wird in Rational Rose RealTime mit Hilfe eines Statecharts beschrieben. Dieses Statechart ist aus Platzgründen nur teilweise in Abbildung 11 dargestellt. Das Verhalten des F-Prozesses wird in SIMATIC Step 7 Graph in der

Ablaufsprache dargestellt. Abbildung 12 zeigt ebenfalls aus Platzgründen nur einen Ausschnitt aus dem gesamten Diagramm.

Wir werden in diesem Beitrag nur den Teil der beiden Diagramme betrachten, der für die Übermittlung der FB-Nachricht *TRS_Out* aus Abbildung 9 im konfliktfreien Fall notwendig ist.

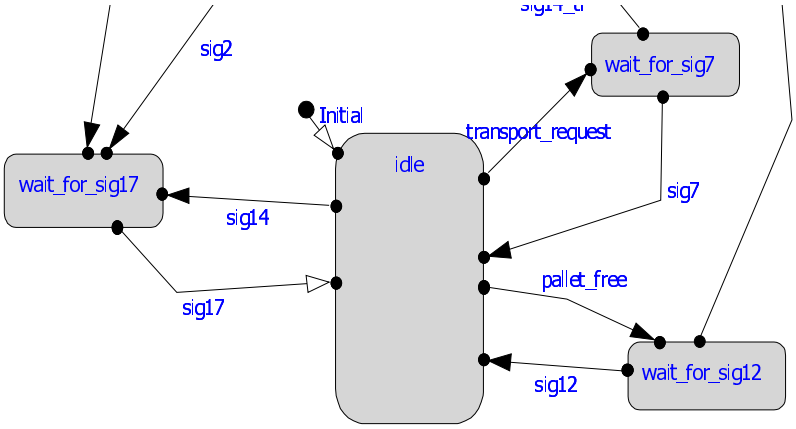

Abbildung 11. Statechart des C-Prozesses (Ausschnitt)

Im FBA-internen Protokoll werden die Nachrichten durch Nummern gekennzeichnet. Zum Beispiel entspricht der FB-Nachricht *TRS_Out* die Nachricht *sig14* für den C-Prozess (Abbildung 11) bzw. *sig14_Out* für den F-Prozess (Abbildung 12). Die Nachrichten mit den Nummer 15 und 17 werden zur Synchronisation zwischen beiden Prozessen des FBA bei der Übermittlung von *TRS_Out* verwendet. Der F-Prozess verwendet weiterhin die Eingangsvariable *OK_In*, um eine Bestätigung für die erfolgreiche Übermittlung einer Nachricht durch das RS232-Gateway zu bekommen.

Werden keine Nachrichten übermittelt, befindet sich der C-Prozess im Zustand *idle* (Abbildung 11) und der F-Prozess im Schritt *idle* (Abbildung 12). Sendet also das Transportsystem *TRS_Out*, schaltet im konflikt-freien Fall die Transition *T1*. Im Schritt *trs_send_14* serialisiert der F-Prozess die aus *TRS_Data_Out* übernommenen Daten (mit der Funktion *convert_sig14_data*) und schreibt sie in den Ausgangsdatenbereich des RS232-Gateways. Dieses wird durch das Setzen von *sig14_Out* dazu aufgefordert, die Daten zu senden. Nach erfolgtem Senden schaltet *T2* und der F-Prozess wartet auf die Nachricht 15 vom C-Prozess. Der C-Prozess führt nach Erhalt der Nachricht 14 die mit der Transition *sig14* verbundene Aktion aus. In dieser Aktion werden die empfangenen Daten zunächst deserialisiert und einer Instanz der Klasse *Trans_resp_Data* zugewiesen. Damit ist die Spezifikation aus Abbildung 9 bis Zeile 19 erfüllt. In der Aktion wird weiterhin die Nachricht *TR_response* an das Roboter-Capsule gesendet und zum Schluss die Nachricht 15 an den F-Prozess. Damit wird dem F-Prozess signalisiert, dass Zeile 20 aus Abbildung 9

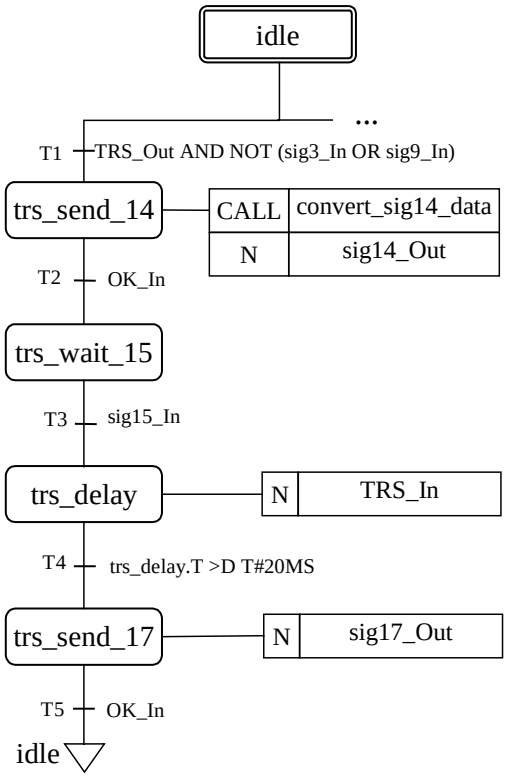

Abbildung 12. Ablaufsprache-Diagramm für F-Prozess

erfüllt wurde und nun *TRS_In* des Transportsystems für 20ms auf *TRUE* gesetzt werden muss (Zeilen 21 bis 23 aus Abbildung 9). Am Ende sendet der F-Prozess die Nachricht 17 an den C-Prozess. Danach gehen beide Prozesse wieder in *idle* über und sind zur Übermittlung einer neuen Nachricht bereit.
Die Umsetzung der Nachrichten Transport_request und Pallet_free geschieht auf ähnliche Art und Weise mit dem Unterschied, dass die Übertragungsrichtung vom Capsule zum Funktionsbaustein verläuft.

5 Zusammenfassung

In diesem Artikel haben wir gezeigt, wie man schon in der Entwurfsphase eines Systems Kommunikationsbeziehungen zwischen Funktionsbausteinen und Capsules mit Hilfe von Funktionsbausteinadaptern modellieren kann. Für die Implementierung eines FBA muss man zunächst ein FBA-internes Kommunikationsprotokoll entwickeln, das auch die notwendige FBA-interne Synchronisation zwischen F- und C-Prozess berücksichtigt.
In unserem Beispiel wurde das Verhalten des C-Prozesses durch ein Statechart und das Verhalten des F-Prozesses durch die Ablaufsprache beschrieben. Es zeigte sich, dass sich beide Darstellungsformen in ihrer jeweiligen Software-Umgebung (nach IEC 61131-3 oder nach UML) für diese Aufgabe eignen.
Als nachteilig erwies sich, dass in der Version des von uns genutzten Ablaufsprache-Tools [4] keine benutzerdefinierten Datentypen für Schnittstellenvariablen verwendet werden konnten. Wir mussten deshalb auf eine Kombination aus globaler Variable und einer Funktion (siehe *convert_sig14_data* aus Abschnitt 4) zurückgreifen. In dem von uns eingesetztem UML-Tool [3] mussten wir plattformabhängige Windows-API Funktionen benutzen, um über die serielle Schnittstelle kommunizieren zu können. Hier wäre ein vom Hersteller mitgeliefertes plattformunabhängiges Protokoll auf Port-Ebene wünschenswert, so dass bei der Code-Generierung für andere Plattformen (z.B. für Echtzeit-Betriebssysteme) an dieser Stelle keine Anpassung notwendig ist.

6 Literatur

[1] Programmable controllers, IEC 61131

[2] B. Selic, J. Rumbaugh, “Using UML for Complex Real-Time Systems”, http://www.rational.com/products/rosert/whitepapers.jsp

[3] Rational Software Corp., "Rational Rose RealTime Users Guide"

[4] SIEMENS AG, “Step 7 Graph 5 Users Guide”

[5] T. Heverhagen, R. Tracht, “Integrating UML-RealTime and IEC 61131-3 with Function Block Adapters”, Proc. of ISORC 2001, May 2-4, 2001, IEEE Computer Society. S. 395-402, http://isorc2001.cs.uni-magdeburg.de/

[6] T. Heverhagen, R. Tracht, “Implementing Function Block Adapters”, Proc. of OMER-2, May 2001, Herrsching a. Ammersee, Report Nr. 2001-03, University of the Federal Armed Forces Munich. S. 11-18
http://inf2-www.informatik.unibw-muenchen.de/GROOM/OMER-2/index.html