Fortschritt-Berichte VDI

VDI

Reihe 8

Mess-,
Steuerungs- und
Regelungstechnik

Nr. 1256

Dipl.-Inf. David Kampert,
Stuttgart

Operative Verwendung merkmalbasierter Information in der Automatisierung

Lehrstuhl für
Prozessleittechnik
der RWTH Aachen

Operative Verwendung merkmalbasierter Information in der Automatisierung

Von der Fakultät für Georessourcen und Materialtechnik der
Rheinisch-Westfälischen Technischen Hochschule Aachen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

genehmigte Dissertation
vorgelegt von **Dipl.-Inf.**

David Kampert

aus Diepholz

Berichter: Univ.-Prof. Dr.-Ing. Ulrich Epple
Univ.-Prof. Dr.-Ing. Christian Diedrich

Tag der mündlichen Prüfung: 06. März 2017

Fortschritt-Berichte VDI

Reihe 8

Mess-, Steuerungs- und Regelungstechnik

Dipl.-Inf. David Kampert, Stuttgart

Nr. 1256

Operative Verwendung merkmalbasierter Information in der Automatisierung

Lehrstuhl für Prozessleittechnik der RWTH Aachen

Kampert, David
Operative Verwendung merkmalbasierter Information in der Automatisierung
Fortschr.-Ber. VDI Reihe 8 Nr. 1256. Düsseldorf: VDI Verlag 2017.
134 Seiten, 22 Bilder, 10 Tabellen.
ISBN 978-3-18-525608-0, ISSN 0178-9546,
€ 52,00/VDI-Mitgliederpreis € 46,80.

Für die Dokumentation: Automatisierung – Industrie 4.0 – IEC 61131 – SPS – Merkmale – Abfragesprache – Relationale Algebra – Dienst

In dieser Arbeit wird ein neuartiges Konzept für die Kommunikation technischer Merkmale vorgestellt. Das Konzept erlaubt industriellen Automatisierungssystemen, Informationen über technische Merkmale mittels IEC 61131-konformer Programmierung von Fremdsystemen, beispielsweise IT-Systemen der MES-Ebene, abzufragen. Durch die Verfügbarkeit dieser Information können flexible Produktionsanlagen deutlich leichter realisiert werden. Grundlage dieser Arbeit ist der aktuelle Stand von Wissenschaft, Technik und industriell angewandten Normen und Standards. Im Detail sind dies theoretische Grundlagen zu Merkmalmodellen und Informationssystemen sowie die praktische Anwendung von Merkmalmodellen, Abfragesprachen, Software-Systemen im industriellen Umfeld und die Programmierung von Automatisierungssystemen. Ausgehend davon werden Anforderungen an das zu entwickelnde Konzept und die Implementierung und Integration abgeleitet. Die Implementierung entsprechender Abfragen in IEC 61131-kompatiblen Automatisierungssysteme wird erläutert. Die Integration in das IT-Umfeld einer industriellen Produktionsanlage wird dabei ebenso betrachtet wie die interne Softwarearchitektur. Die Arbeit schließt mit Anwendungsbeispielen und einer kritischen Diskussion des Konzepts.

Bibliographische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet unter http://dnb.ddb.de abrufbar.

Bibliographic information published by the Deutsche Bibliothek
(German National Library)
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie (German National Bibliography); detailed bibliographic data is available via Internet at http://dnb.ddb.de.

D82 (Diss. RWTH Aachen University, 2017)

Als Manuskript gedruckt. Printed in Germany.
ISSN 0178-9546
ISBN 978-3-18-525608-0

Vorwort

Diese Arbeit entstand während meiner Zeit als wissenschaftlicher Mitarbeiter des Lehrstuhls für Prozessleittechnik der RWTH Aachen. Ich bedanke mich bei Herrn Professor Dr.-Ing. Ulrich Epple dafür, dass er mir dieses Unterfangen nicht nur ermöglichte, sondern durch seine Führung des Lehrstuhls auch zu einer angenehmen, lehrreichen, konstruktiven und schönen Zeit gemacht hat. Ich danke außerdem Professor Dr.-Ing. Christian Diedrich für die freundliche Übernahme der Rolle des Zweitgutachters.

Großen Anteil an dieser Arbeit hat auch das gesamte Team des Lehrstuhls, zu dem ich über fünf Jahre gehören durfte. Ein herzliches „Dankeschön“ an dieses tolle Team!

David Kampert

Inhaltsverzeichnis

Kurzfassung

In dieser Arbeit wird ein neuartiges Konzept für die Kommunikation technischer Merkmale vorgestellt. Das Konzept erlaubt industriellen Automatisierungssystemen, Informationen über technische Merkmale mittels IEC 61131-konformer Programmierung von Fremdsystemen abzufragen. Die Arbeit erläutert die theoretischen Grundlagen, das Gesamtkonzept und Aspekte der Integration für die industrielle Automatisierung.

Geräte und Systeme im industriellen Umfeld werden vermehrt kommunikationsfähig und mit Netzwerken verbunden. Diese Konnektivität wird meist für den Zugriff auf Automatisierungssysteme von außen benutzt, aber auch die Automatisierungssysteme selbst können von den dadurch zugänglichen Daten profitieren, indem sie operativ notwendige oder nutzenbringende Information durch das Netz von Fremdsystemen abfragen. Beispielsweise kann eine Anlagensteuerung die notwendige Information zur Herstellung einer Produktvariante selbst und zum richtigen Zeitpunkt erfragen, ohne aktiv von außen mit der Information versorgt werden zu müssen. Solche Abfragen sind nach heutigem Stand der Technik aber aufwändig einzurichten, zu warten und nicht flexibel. Die Vernetzung birgt also großes Potenzial für die Automatisierung, das Verhältnis von Aufwand und Nutzen ist aber ungünstig.

Eine große Vereinfachung wäre eine Standardisierung der von Automatisierungssystemen ausgehenden Kommunikation. Während in technischer Hinsicht bereits Lösungen existieren, fehlt es in semantischer Hinsicht an Standards. Die Bedeutung von Information aus einem Fremdsystem, beispielsweise Information über ein Produktmerkmal, muss vorab bekannt sein, was mit steigender Anzahl vernetzter Systeme schwieriger wird. Weil ein großer Teil der Kommunikationsinhalte, die für ein Automatisierungssystem operativ nutzbar sind, technische Merkmale betrifft, sind diese der Ausgangspunkt für das in dieser Arbeit vorgestellte Konzept.

Heute sind umfangreiche Merkmaldefinitionen durch Normen und Standards verfügbar (z.B. IEC 61360 und eCl@ss). In der Wissenschaft existiert außerdem eine klare Vorstellung davon, wie das Prinzip der Modellierung durch Merkmale grundsätzlich funktioniert. Für die operative Nutzung dieser Daten und Modelle in der Automatisierung gibt es aber kein Konzept.

Grundlage dieser Arbeit ist der aktuelle Stand von Wissenschaft, Technik und industriell angewandten Normen und Standards. Dieser wird in den ersten Kapiteln der Arbeit in Hinblick auf das Gesamtkonzept vorgestellt und erläutert. Im Detail sind dies theoretische Grundlagen zu Merkmalmodellen und Informationssystemen sowie die praktische Anwendung von Merkmalmodellen, Abfragesprachen, Software-Systemen im industriellen Umfeld und in die Programmierung von Automatisierungssystemen. Ausgehend davon werden Anforderungen an das zu entwickelnde Konzept und die Implementierung und Integration abgeleitet.

Der erste Schritt zur Lösung ist die Definition einer Abfragesprache für technische Merkmale, die sich auf ein vorhandenes allgemeines Metamodell für Merkmale bezieht. Die Abfragesprache hat die relationale Algebra als formale Basis. Die Operationen

der Sprache bestehen aus Grundoperationen, die durch theoretische Überlegungen und technische Umsetzung motiviert sind sowie aus erweiterten Operationen, die durch die praktische Anwendung motiviert sind und auf den Grundoperationen aufbauen.

Anschließend wird die Integration entsprechender Abfragen in IEC 61131-kompatible Automatisierungssysteme erläutert. Die Operationen werden als Typen von Funktionsbausteinen spezifiziert, so dass Abfragen von Merkmalinformation durch Funktionsbausteine programmiert werden können. Schnittstellen und Ausführungssemantik werden spezifiziert.

Letztlich werden der Entwurf und die Integration eines Servers zur Verarbeitung dieser Abfragen diskutiert. Die Integration in das IT-Umfeld einer industriellen Produktionsanlage wird dabei ebenso betrachtet wie Grundzüge der internen Softwarearchitektur. Die Arbeit schließt mit Anwendungsbeispielen und einer kritischen Diskussion des Konzepts.

1. Einleitung

1.1. Motivation

Durch die Vernetzung von Automatisierungssystemen industrieller Produktionsanlagen mit IT-Systemen und dem Internet wird es in naher Zukunft große Veränderungen in der Industrie, der Wirtschaft und der Gesellschaft geben [1, 77]. Durch die Verfügbarkeit von Information werden vorhandene Systeme ihre Aufgaben besser erfüllen können und es werden vollkommen neuartige technische Lösungen und Geschäftsmodelle entstehen. Die Systeme zur Organisation und Steuerung der Produktion sind dabei Mittel- und Ausgangpunkt der Veränderungen. Diese Arbeit beschäftigt sich mit den neuen Möglichkeiten für die Automatisierungstechnik. Genauer gesagt wird eine Technologie eingeführt, die es speicherprogrammierbaren Steuerungen erlaubt, mittels der Vernetzung auf Daten technischer Merkmale zuzugreifen. Durch diese neu verfügbare Informationsquelle können viele Automatisierungsaufgaben besser erfüllt werden und es entstehen auch gänzlich neue Anwendungsmöglichkeiten.

Speicherprogrammierbare Steuerungen, abgekürzt SPS, sind Systeme für die programmierbare Automatisierung von technischen Anlagen, beispielsweise industriellen Produktionsanlagen. Die Zielsetzung für Entwickler und Anwender einer SPS war und ist es, dass mit dem System Vorgänge der realen Welt gemessen, gesteuert und geregelt werden können – möglichst effektiv, präzise, umfangreich, kostengünstig und aufwandsarm. Dem Programmierer einer SPS stehen dafür die notwendigen Mittel zur Verfügung, d.h. die technische Anbindung der Feldgeräte und, getrennt davon, die Werkzeuge zur Implementierung und Ausführung von Berechnungen und Prozeduren. Für Mess-, Regel- und Steueraufgaben sind diese Mittel geeignet. Zur Ausnutzung der Vorteile einer vernetzten Infrastruktur ist das jedoch nicht ausreichend. Das ist ein Nachteil, denn es befindet sich in vernetzten Systemen im Umfeld industrieller Produktionsanlagen viel Information, die prinzipiell für eine bessere und flexiblere Funktion einer SPS genutzt werden könnte: Informationen zu Merkmalen von Produktionsplanung, Aufträgen, Produkten, Rohstoffen, Geräten der Anlage, Kosten oder Logistik sind nur einige Beispiele. Trotz heute oder zukünftig vorhandener Vernetzung ist diese Information für eine SPS kaum erreichbar, denn ein Programmiermittel zum gezielten Suchen und Erfragen von Information gibt es nicht. Für den ursprünglichen Einsatzzweck von SPS zum Messen, Steuern und Regeln muss keine Information gesucht werden, daher gibt es hierzu kein nativ vorhandenes Konzept. Als Lösung bieten einige Softwarehersteller spezielle Bibliotheken an, mit deren Hilfe Information aus SQL-Datenbanken oder Excel-Dateien abgefragt bzw. herausgesucht werden kann. Dies sind aber jeweils technische Speziallösungen; die logische Abfrage einer Information ist dann untrennbar mit der Technologie und Strukturierung der Datenquelle verbunden. Ein selbständiges Konzept, das die technische Kommunikation von der logischen Informationsabfrage und -verwendung trennt, gibt es nicht. Das sind Ausgangspunkt und Motivation dieser Ar-

beit. Ziel ist die Spezifikation einer allgemeinen Abfragesprache für Merkmalinformation, die von einer SPS genutzt werden kann.

Diese Arbeit kann aus drei Perspektiven betrachtet werden.

1.1.1. Die wissenschaftliche Perspektive

Dieser Arbeit liegt die These zugrunde, dass es Grundprinzipien zur Strukturierung von Information gibt. Im hier betrachteten Anwendungsumfeld der industriellen Produktion gibt es zahlreiche Modelle, die Aspekte von Produktionsanlagen, Geräten, Produkten, Produktionsabläufen, Produktionsplänen usw. abbilden. Diese Modelle liegen heute zunehmend in IT-Systemen vor und besitzen oft auch ein dokumentiertes Meta-Modell. Die Modelle vermitteln vier Arten von Information:

- Merkmalinformation beschreibt charakteristische Eigenschaften von Objekten, die für das besitzende Objekt als Ganzes gelten und mit einfachen Werten belegt werden können. Datenblätter vermitteln hauptsächlich Merkmalinformation.
- Strukturinformation erklärt, aus welchen Objekten das modellierte Gesamtobjekt besteht und wie diese Objekte zueinander in Beziehung stehen. Ein R&I-Fließbild vermittelt hauptsächlich Strukturinformation.
- Zeitbezogene Information beschreibt die unterschiedlichen Zustände, in denen ein Objekt war, (in Zukunft) sein soll oder sein kann. Das kann durch explizite Beschreibung der Zustände (z.B. bei der Modellierung eines Produktlebenszyklus) oder durch die Beschreibung der Dynamik (z.B. Differenzialgleichungen) geschehen.
- Bildhafte Information dient der Nachbildung der Sinneswahrnehmung des modellierten Gegenstands, typischerweise der optischen Erscheinung. 3D-CAD-Modelle vermitteln bildhafte Information.

In den meisten gebräuchlichen Modellen werden diese Arten von Information miteinander vermischt und die Modelle können aufeinander aufbauen. Merkmalinformation ist in beinahe allen Modellen vorhanden, allein schon, um den modellierten Gegenstand und ggf. dessen Bestandteile anhand von Merkmalen wie „Name“ zuordnen zu können. Der Zugriff auf Merkmalinformation ist daher für besonders viele Modellarten relevant.

Von wissenschaftlicher Seite wurde das große Potenzial, das mit der Beherrschung von Merkmalinformation einhergeht, identifiziert (s. z.B. [29, 34, 63, 67]). Ein Metamodell für Merkmale bzw. eine einheitliche Sicht auf Modelle mit Merkmalinformation wurde von Mertens veröffentlicht [54]. Die Frage danach, wie dieses Modell operativ – beispielsweise von Automatisierungssystemen – genutzt werden kann, ist aber bisher ohne Antwort. In vielen Fällen wird Merkmalinformation nicht lokal vorliegen, sondern von Fremdsystemen abgefragt werden müssen. In dieser Arbeit wird daher systematisch und ausgehend vom Stand von Wissenschaft und Technik ein Konzept für die Abfrage dieser Information spezifiziert, das auf das allgemeine Modell für Merkmale anwendbar und die Anwendung in der Automatisierungstechnik zugeschnitten ist. Es vervollständigt so das vorhandene Metamodell aus funktionaler und operativer Sicht der Automation.

1.1.2. Die pragmatische Perspektive

Durch die Vernetzung industrieller Produktionssysteme wird die Menge verfügbarer Information zunehmend größer. Grundsätzlich bietet diese Verfügbarkeit von Information das Potenzial, den Anlagenbetrieb und die Produktionsprozesse selbst zu verbessern bzw. sogar neuartige Prozesse zu ermöglichen. Voraussetzung dafür ist aber, dass die Information nicht nur verfügbar ist, sondern die Informationsmenge und die Abwicklung der Kommunikation auch beherrschbar sind. Die Größe und Dynamik der Netze erschweren das Auffinden der jeweils gesuchten Information. Das ist besonders für Automatisierungssysteme eine Herausforderung, weil sie über vergleichsweise geringe Hardwareressourcen verfügen und nicht zu beliebigen Zeitpunkten zwecks Anpassung an Veränderungen neu programmiert werden können. Folgendes Praxisbeispiel illustriert dieses Problem:

In einer Raffinerie soll der Arbeitszustand von Erdölpumpen durch die steuernde SPS überwacht werden. Dafür werden spezielle Funktionsbausteine in der SPS verwendet, die die Betriebsparameter, Zustände und Kenndaten der Pumpen als Eingänge haben. Zusätzlich benötigen die Bausteine aufgrund physikalischer Gesetzmäßigkeiten Informationen über das geförderte Medium, hier also das Erdöl. Die SPS benötigt beispielsweise Information darüber, welche Dichte das Erdöl aus der Probe hat, die aus dem Tank stammt, aus dem derzeit gepumpt wird. Dieses Merkmal ist je nach verwendeter Erdölsorte verschieden und liegt im Laborinformationssystem vor. Gleichzeitig kennt das MES notwendige Details zum aktuellen Arbeitsauftrag wie den genutzten Tank. Die Komplexität der Datenstrukturen und Netze (Firewalls, Regeln zur Sicherheit etc.) erschwert die direkte Beschaffung der benötigten Information, außerdem führen Veränderungen der Informationsquellen durch Migration oder Updates zu erzwungenen Anpassungen der SPS.

Idealerweise könnte die SPS in dieser Situation die benötigte Information selbst erfragen, ohne dass eine Punkt-zu-Punkt-Verbindung zu den Quellsystemen der Information erstellt und gewartet werden muss. Dass die SPS selbst die benötigte Information erfragen kann, ist derzeit ein eher ungewöhnlicher Weg zur Lösung dieser Art von Aufgaben, weil üblicherweise die SPS „von außen“ mit der benötigten Information versorgt wird. Das liegt aber nicht daran, dass die technischen Möglichkeiten für die aktive Datenabfrage fehlen würden oder dass der Weg grundsätzlich organisatorisch komplizierter wäre. Es gibt schlicht kein effektiven Konzept. Eine sehr ähnliche Idee ist unter der Bezeichnung „Enterprise Information Integration“ ist im Bereich der IT-Systeme bereits seit Anfang der 1990er Jahre bekannt [35], wo der Anwendungsbereich aber bei großen Datenbanken und komplexen Datenstrukturen liegt. Somit gibt es für die technische Umsetzung in großem Maßstab bereits Beispiele, aus Sicht der Automatisierung fehlt aber eine Lösung, die sich auf die Abbildung von Merkmalen konzentriert und die auf die domänenspezifischen Anforderungen zugeschnitten ist. Diese Arbeit liefert das Konzept für eine solche Lösung.

1.1.3. Die strategische Perspektive

Für die eingangs des Kapitels erwähnten Veränderungen in der Industrie, die z.T. begonnen haben und noch in Zukunft erwartet werden, spielt die Automatisierungstechnik eine zentrale Rolle. Umso erstaunlicher ist, dass in vielen Vorschlägen, wie zukünfti-

ge Automatisierung „intelligenter“ und „flexibler“ gemacht werden kann, die eigentlichen Automatisierungssysteme, die reale Prozesse messen, steuern und regeln, sich überhaupt nicht verändern. Die Änderungen beziehen sich stattdessen auf Software-Systeme im Umfeld der Automatisierungssysteme, die mehr und besser als zuvor Daten sammeln und auswerten, um damit dann die Automatisierungssysteme zu parametrieren, konfigurieren oder programmieren. Automatisierungssysteme sind somit keine aktiven Komponenten in der vernetzten virtuellen Welt. Es ist daher möglich, dass die Automatisierungstechnik als Disziplin von den Ergebnissen der „Industrie 4.0“, dem „Industrial Internet“ und von „Cyber-Physical Production Systems“ trotz ihrer zentralen Rolle nur wenig profitiert. Die Motivation dieser Arbeit aus strategischer Perspektive ist daher zu zeigen, wie Automatisierungssysteme auf einfache und effektive Art von der Vernetzung mit IT-Systemen profitieren können und sich dadurch neue Anwendungsmöglichkeiten eröffnen. Es entsteht eine neue Art von Softwaresystem, das (anders als vorhandene Systeme) keine neue Information erzeugt, sondern nur vorhandene Information in einer Art organisiert, die für die Automatisierungstechnik einen direkten Nutzen erzeugt.

1.2. Zielsetzung und Idee

Ziel der Arbeit ist die Spezifikation einer allgemeinen Abfragesprache für Merkmalinformation, die von einer SPS operativ genutzt werden kann. Dieses Ziel geht mit der Betrachtung einer Reihe von Nebenbedingungen einher, um die tatsächliche Realisierbarkeit und Einsetzbarkeit des Konzepts zu gewährleisten. Die gesuchte Lösung muss in das technische Umfeld der industriellen Produktion passen, d.h. mit vorhandenen Software-Systemen integrierbar sein und grundsätzlich in einer SPS verwendbar. Außerdem sollte die Lösung innerhalb der SPS kein neu- und andersartiger Fremdkörper sein, damit sie leicht erlernbar und nachvollziehbar ist. Die Einhaltung von Standards und die Orientierung an vorhandenen Programmierkonzepten sind daher unverzichtbar. Weiterhin sollte für einen Anwender klar sein, in welchen Fällen die Abfragesprache eingesetzt werden kann und wo ihre Grenzen liegen. Die Entwicklung neuer Theorie im Bereich der Informationssysteme oder neuer Datenbanktechnologie sind dagegen nicht Ziel der Arbeit.

Der Idee dieser Arbeit gehen einige Grundüberlegungen zur Kommunikation zwischen Automatisierungs- und IT-Systemen voraus. Für das allgemeine Ziel, einer SPS Merkmalinformation zur Verfügung zu stellen, gibt es mehrere denkbare Lösungen (s. Abbildung 1.1). Eine naheliegende Möglichkeit, gezeigt bei ①, ist dass die externen Systeme die jeweils benötigten Daten in die SPS schreiben. Aus technischer Sicht ist das möglich, weil entsprechende Datenschnittstellen wie OPC DA/UA oft verfügbar sind. Aus logischer Sicht funktioniert diese Lösung jedoch häufig nicht, weil die Informationsquelle wissen müsste, welche SPS welche Information benötigt. Im Beispiel der Pumpenüberwachung müsste also das Laborinformationssystem wissen, aus welchem Tank das von einer Pumpe geförderte Rohöl stammt. Diese Information ist im Laborinformationssystem nicht vorhanden. Selbst wenn das der Fall wäre, ist dieser Lösungsweg organisatorisch schwierig: Wenn sich etwas an der Anlage oder der SPS ändert, so dass andere Informationen benötigt werden als zuvor, müsste die Informationsquelle angepasst werden, obwohl sie von der ursprünglichen Änderung eigentlich nicht betroffen ist. Anders-

herum betrachtet muss bei einer Änderung an der Informationsquelle die spezielle Abhängigkeit zur SPS berücksichtigt werden. Solche Abhängigkeiten führen zu erhöhtem Arbeitsaufwand, Abstimmungsbedarf, Verzögerungen und Inkonsistenzen. Letztlich ist Lösung ① daher nicht ideal.

Das führt zu Lösung ②, in der die SPS selbst bei den externen Systemen Information abfragt. Weil IT-Systeme normalerweise nicht darauf ausgerichtet sind, dass Daten von einer SPS aus abgefragt werden, bieten sie im Regelfall auch keine speziellen Schnittstellen dafür an. Die SPS muss sich daher nach den angebotenen Schnittstellen richten, was technisch z.T. durch spezielle Bibliotheken auf mehr oder weniger umständliche Art möglich ist (im Bild verdeutlicht durch die unterschiedlichen Befehle `get` und `rq` zur individuellen Datenabfrage). Auch hier ist aber wieder der organisatorische Aspekt schwierig. IT-Systeme haben meist kürzere Lebenszyklen als Automatisierungssysteme und ändern sich daher während des Betriebs der SPS, die ggf. wegen der laufenden Produktion auch nicht neu programmiert werden kann [52]. Insgesamt bestehen bei Lösung ② also immer noch erhebliche technische und organisatorische Probleme.

In Lösung ③ wird die technische Abhängigkeit durch ein zusätzliches Software-System aufgelöst, das als eine Art Broker zwischen SPS und Datenquellen vermittelt. Dieses System implementiert die auf der jeweiligen Seite vorhandenen Schnittstellen und entkoppelt IT-Systeme und Automatisierungssysteme voneinander. Bei einer geschickten Implementierung können Veränderungen in einem beteiligten System im Voraus eingeplant und gleitende Übergänge geschaffen werden. Es bleibt einzig das Problem, dass das neu eingeführte System wissen muss, welche Information von welcher SPS benötigt wird. Wünschenswert wäre, wenn das Broker-System nur die rein technische „Übersetzung" von Abfragen durchführen müsste, ohne dass hier konfiguriert werden muss, wer welche Information benötigt. Das führt letztendlich zu Lösung ④, in der der vermittelnde Broker für die Automatisierungssysteme eine Schnittstelle anbietet, an der Daten abgefragt werden können. Die Formulierung der Abfrage geschieht in der SPS und sofern andere Daten benötigt werden als zuvor, muss auch nur dort die Abfrage geändert werden. Das vermittelnde System in der Mitte ist eine rein technische Brücke und die IT- und Automatisierungssysteme sind logisch so weit wie möglich entkoppelt. Bei technischen Änderungen muss nur die „Brücke" angepasst werden, bei einer Änderung des Informationsbedarfs einer SPS nur genau diese. Technische Kommunikation und logische Informationsabfrage sind genauso getrennt, wie es auch bei der Signalanbindung für Feldgeräte und der späteren Informationsverarbeitung durch Programmlogik getan wird.

Konzeptionell ist die letzte diskutierte Lösung also zu favorisieren und auch in technischer Hinsicht sind die notwendigen Voraussetzungen bereits erfüllbar. Dadurch, dass ein neues Software-System in einer frei wählbaren Technologie implementiert wird, kann sich die Wahl hierfür nach den benötigten Kommunikationsschnittstellen richten. Die kommunizierte Merkmalinformation ist auch inhaltlich einfach genug, so dass vorerst nicht mit technischen Problemen bei der Datenübertragung zu rechnen ist. Aus der Literatur liegt auch ein allgemeines Modell von Merkmalinformation vor [29, 54], so dass es eine einheitliche und dokumentierte Sicht auf Merkmalinformation gibt. Es fehlt aber noch an den Mitteln, die die allgemeine Abfrage von Merkmalinformation aus einer SPS erlauben und an der „Brückensoftware" zu den merkmalverwaltenden Systemen. Randbedingungen sind, dass Beides in das technische Umfeld passen muss und dass die Formulierung von ausreichend komplexen Abfragen möglich ist – beispielsweise die

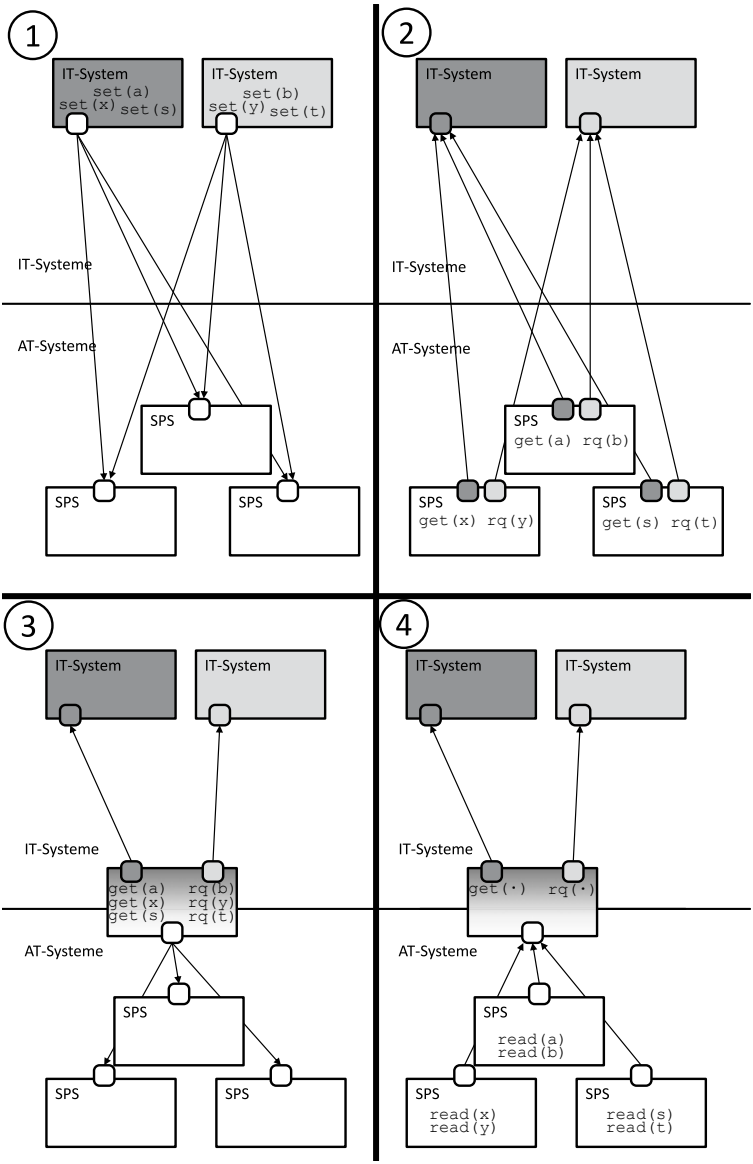

Abbildung 1.1.: Vier Möglichkeiten einer Architektur, in der Automatisierungssysteme Informationen aus IT-Systemen operativ nutzen.

Abfrage der Dichte des Rohöls aus dem Tank, aus dem eine bestimmte Pumpe derzeit Rohöl fördert. Diese Arbeit beschreibt genau diese noch fehlenden Teile des Konzepts.

Abbildung 1.2 illustriert die Grundidee aus praktischer Sicht anhand eines Beispiels. Zu sehen sind eine Datenquelle mit Merkmalinformation, eine SPS mit einem Programm aus Funktionsbausteinen und ein drittes System, das die SPS an eine oder mehrere Datenquellen anbindet und Abfragen von Merkmalinformation ausführt. Der Aufbau des Programms aus Funktionsbausteinen wird auch später in dieser Arbeit weiter verfolgt und begründet, ist hier aber hauptsächlich aus Gründen der Darstellung hilfreich. Der Anwender kann aus vorhandenen Elementen wie den Bausteinen „Suche“ und „Istwert“ eine Abfrage zusammenstellen und dann gezielt (mit dem Baustein „Abfragen“) ausführen lassen. Konkret wird aus DatenquelleX derjenige Merkmalträger mit WertZ für MerkmalA gesucht. Von diesem Merkmalträger wird dann auf das EreignisE hin der Istwert von MerkmalB erfragt. Dass die Abfrage tatsächlich von einem externen System ausgeführt wird, ist für den Anwender nicht relevant und unsichtbar. Das Ergebnis der Abfrage kann als normale Variable weiterverwendet werden, hier dargestellt durch eine Signallinie hin zu einem leeren Funktionsbaustein. Der Aufbau komplexer geschachtelter Abfragen und die Einbindung mehrerer Datenquellen sind ebenfalls möglich. Im Prinzip handelt es sich um eine mit Mitteln der SPS formulierte Datenabfrage ähnlich wie eine SQL-Query im PC-Bereich, aber ohne Annahmen über die technische Speicherung der Daten. Stattdessen liegt nur die Annahme zugrunde, dass die Abfrage Merkmalinformation betrifft. Die Lösung ist daher allgemein immer dann anwendbar, wenn Merkmalinformation erfragt wird.

1.3. Aufbau der Arbeit

Die in Abschnitt 1.2 geschilderte Idee und Zielsetzung wirft eine ganze Reihe von Fragen auf, z.B.:

- Welche Modelle zur Abbildung von Merkmalinformation gibt es in der Wissenschaft?
- Welche Modelle zur Abbildung von Merkmalinformation gibt es in der Normung und der industriellen Praxis?
- In welchen Software-Systemen im Bereich industrieller Produktion liegt verwendbare Merkmalinformation vor?
- Welche Möglichkeiten bietet ein Automatisierungssystem zur Formulierung und Verarbeitung von Abfragen von Merkmalinformation?
- Auf welchen normativen Grundlagen für die SPS-Programmierung kann aufgebaut werden?
- Wie kann eine Abfragesprache systematisch hergeleitet werden?
- Welche Abfragen sind mit dieser Sprache möglich und gibt es Einschränkungen?
- Wie wird eine Abfrage verarbeitet?

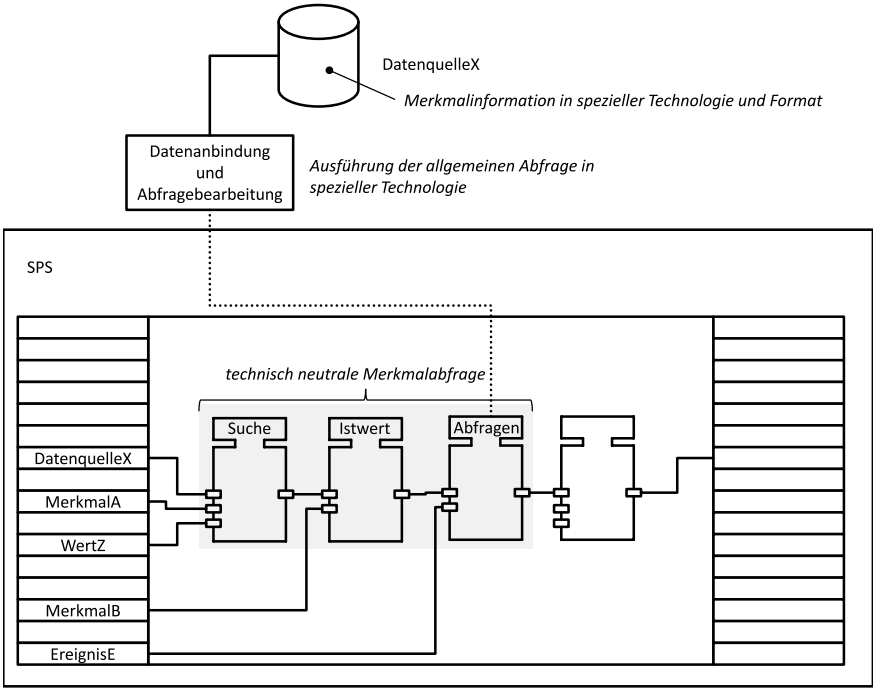

Abbildung 1.2.: Idee der Arbeit anhand eines Beispiels.

- Wie sollte ein System zur Verarbeitung von Abfragen aufgebaut und mit vorhandenen Systemen integriert werden?

Diese Fragen werden in den folgenden Kapiteln Schritt für Schritt beantwortet.

Abbildung 1.3 zeigt den Aufbau dieser Arbeit in Form eines V-Modells. Im folgenden Kapitel 2 werden die theoretischen Grundlagen behandelt, auf denen später eine Lösung aufgebaut wird. Die wesentlichen Grundlagen sind die Modellierung von Merkmalen und die Theorie von Informationssystemen. In Kapitel 3 wird dann auf derzeitig benutzte Technologien im hier adressierten Anwendungsgebiet industrielle Automatisierung eingegangen. Die gebräuchlichsten Merkmalmodelle und Softwaresysteme sowie Grundlagen der Programmierung von SPS werden erläutert. Kapitel 4 analysiert anschließend, wo im aktuellen Stand der Technik der Bedarf an neuen Konzepten und Lösungen zur Erreichung der Zielsetzung besteht und wie die Grundlagen aus Kapitel 2 angewendet werden können. Anforderungen an ein Lösungskonzept werden formuliert, so dass dann in Kapitel 5 eine konzeptuelle Lösung als Antwort auf diese Anforderungen hergeleitet werden kann. Dies sind der Kern und Hauptbeitrag der Arbeit. Ein implementierter Prototyp wird in Kapitel 6 dokumentiert. Kapitel 7 diskutiert den Beitrag dieser Arbeit kritisch.

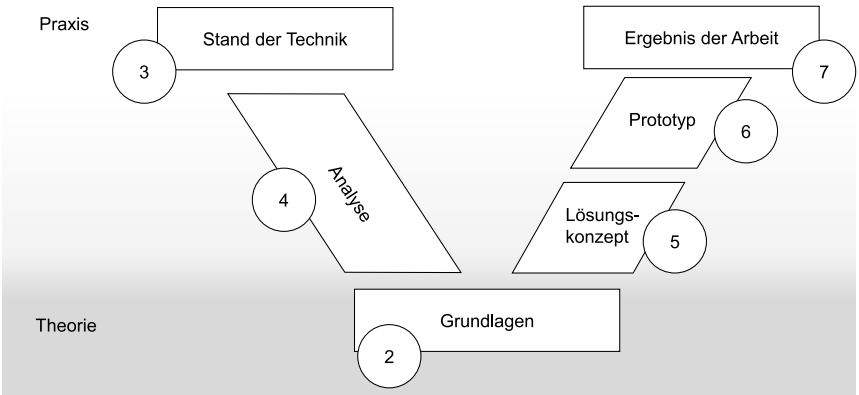

Abbildung 1.3.: Aufbau der Arbeit in Form des V-Modells.

2. Grundlagen

2.1. Merkmale

Merkmale sind klassifizierte Eigenschaften eines Systems, deren jeweilige Ausprägung einen einfachen Wert annimmt [32].

Merkmale sind ein grundlegendes Mittel zur Beschreibung von Systemen und Objekten, das jeder Mensch intuitiv anwendet. In dieser Arbeit steht jedoch die Verarbeitung von Merkmalen durch technische Systeme im Mittelpunkt, so dass Merkmale als Mittel zur Modellierung betrachtet werden, das zur automatisierten Verwendung einer exakten Beschreibung bedarf.

Wenn technische Systeme modellhaft beschrieben werden und diese Modelle durch (zumeist andere) technische Systeme verarbeitet werden, sind Merkmale praktisch immer ein Bestandteil dieser Modelle. Man denke allein nur daran, dass die Identifikation des modellierten Objekts in den allermeisten Fällen durch Merkmale wie „Name“ oder „Identifikationsnummer“ geschieht. Diese Tatsache hat zwei Konsequenzen: Erstens werden Merkmale als gewissermaßen „natürliche“ Bestandteile von Modellen betrachtet, die nicht weiter erklärt sind. Die einfache Benennung eines Merkmals und eines Wertes („Prozessorkerne: 4“) bedarf meist auch keiner weiteren Erklärung – allerdings nur dann, wenn das Modell in einem vorab bekannten Kontext verwendet wird. Anderenfalls, insbesondere bei der Interoperation unterschiedlicher Systeme, ist diese Information unzureichend. Die zweite Konsequenz ist, dass Merkmale in Modellen aller Anwendungsdomänen vorkommen und oft auch das einzige gemeinsame Modellierungsprinzip sind. Beispielsweise sind ein technischer Plan einer Produktionsanlage und die Darstellung der Anlage als Kostenstelle vollkommen unterschiedliche Modelle, die aber durch das Merkmal „Name“ derselben Anlage einander zugeordnet werden können. Die Verknüpfung unterschiedlicher Modelle über gemeinsame Merkmale bietet daher enormes Potenzial für die automatisierte Verwendung von Modellen. Dieses Potenzial lässt sich aber nur dann ausschöpfen, wenn bei der Verknüpfung der Modelle dasselbe Metamodell zur Modellierung von Merkmalen zugrundegelegt wird.

2.1.1. Metamodell zur Modellierung von Merkmalen

In diesem Abschnitt wird das Metamodell zur Modellierung von Merkmalen zusammengefasst. Im Folgenden wird dabei abkürzend vom „Merkmalmodell“ gesprochen, obwohl es sich eigentlich um Modellierung auf der Modell- und auf der Metamodellebene handelt. Der Aufbau des Merkmalmodells ist der Arbeit von Mertens [54] entnommen, in der es ausführlich motiviert und erläutert wird. Die Terminologie orientiert sich dagegen an [29].

Merkmalträger

Das Merkmalmodell wird zur Modellierung individueller Gegenstände der physischen Welt oder der Informationswelt verwendet. Zu diesen Gegenständen können beliebige weitere Modelle existieren. Über Existenz und Art dieser anderen Modelle, und auch über den modellierten Gegenstand, werden durch das Merkmalmodell keine weiteren allgemeingültigen Aussagen gemacht. Über den Gegenstand ist nur bekannt, dass er Träger von Merkmalen ist, woraus sich die Bezeichnung „Merkmalträger“ ableitet. Diese Bezeichnung ist in jedem Fall zutreffend.

Der eigentliche Merkmalträger ist der modellierte Gegenstand selbst, nicht etwa das Modell des Gegenstands. Er selbst ist Besitzer der Merkmale, unabhängig davon, ob es ein merkmalbasiertes Modell des Merkmalträgers gibt oder nicht. Beispielsweise hat ein Produkt stets Herstellungskosten, auch wenn diese im Einzelfall nicht immer erfasst werden. Welche Merkmale einem Merkmalträger zugeschrieben werden ist aber eine Designentscheidung in der Modellierung und hängt wesentlich von der Modellverwendung ab. Allein am Merkmalträger ist das Vorhandensein eines Merkmals daher nicht erkennbar. Das Merkmal erhält dadurch eine vermittelnde Rolle zwischen Merkmalträger und seinem Modell: Es wird in der Modellierung definiert, die Ausprägung ist aber Eigenschaft der modellierten Gegenstands.

Die Unterscheidung zwischen dem realen Merkmalträger und dessen modellhafter Abbildung kann im Einzelfall ein wichtiger Aspekt sein. Das gilt insbesondere dann, wenn es keine eins-zu-eins Abbildung zwischen realem Merkmalträger und dessen Modell gibt. Daher wird an dieser Stelle explizit darauf hingewiesen, dass dieser Unterschied existiert. Zur sprachlichen Vereinfachung wird in dieser Arbeit aber dort, wo es unmissverständlich ist, abkürzend der Begriff „Merkmalträger“ auch für das einzelne Modell des Merkmalträgers verwendet. Beispielsweise beinhaltet eine Inventardatenbank eigentlich Modelle der inventarisierten Merkmalträger, es ist aber im Allgemeinen unproblematisch und einfacher, wenn auch sprachlich ungenau, von den enthaltenen Merkmalträgern zu sprechen.

Ausprägung von Merkmalen

Sofern einem Merkmalträger ein Merkmal zugeordnet wird, kann dessen Ausprägung durch Betrachtung des Merkmalträgers direkt oder indirekt bestimmt werden. Der indirekte Schluss auf ein Merkmal kann erforderlich sein, wenn es für das Merkmal keine anwendbaren Messmethoden gibt oder auch wenn die Ausprägung nur in der Informationswelt existiert und überhaupt nicht physisch präsent ist. Beispielsweise ist der finanzielle Wert eines Gegenstands nicht direkt am Gegenstand messbar, trotzdem würde der Gegenstand als Besitzer des Merkmals betrachtet.

Die Ausprägung eines Merkmals kann im Verlauf des Lebenszyklus eines Merkmalträgers variieren. Viele Eigenschaften eines Merkmalträgers, die im Sinne des Merkmalmodells als Merkmale gelten, können deshalb auch als Zustände des Merkmalträgers angesehen werden. Entscheidend ist hier die Verwendung des merkmalbasierten Modells: Sofern während des Zeitraums, in dem das Modell angewendet wird, die Ausprägung als stabil angesehen werden kann, kann eine Eigenschaft als Merkmal verwendet werden. Zum Beispiel ist die Füllmenge eines Transportbehälters allgemein kein Merkmal des Behälters, weil sie sich während des Lebenszyklus eines Transportbehäl-

ters ändert. Aus Sicht einer Logistiksoftware wird aber unter Umständen nur ein Zeitabschnitt betrachtet, in dem die Füllmenge konstant ist. Im Modell der Logistiksoftware kann der Behälter daher das Merkmal „Füllmenge“ besitzen.

Ausprägungsaussagen

Wenn anhand des Merkmalmodells ein Merkmalträger modelliert wird, dann sollen normalerweise auch Informationen über die Ausprägungen der Merkmale im Modell hinterlegt werden. Diese Information über Ausprägungen darf aber nicht mit den tatsächlichen Ausprägungen verwechselt werden, die ja nur am Merkmalträger selbst messbar sind. Daher sind im Modell nur Aussagen über Ausprägungen vorhanden, sogenannte „Ausprägungsaussagen“. Ausprägungsaussagen setzen ein Merkmal in eine bestimmte Beziehung zu einem Wert, gegebenenfalls natürlich mit Angabe einer Maßeinheit. Zum Beispiel kann eine Ausprägungsaussage sein, dass ein Wert x der gemessene Wert des Merkmals ist, der simulierte Wert, der optimale Wert oder der geforderte Wert. Wenn eine Ausprägungsaussage einem bestimmten Merkmalträger zugeordnet wird, dann bezieht sich die Aussage auf das entsprechende Merkmal von genau diesem Merkmalträger und lässt keine Schlüsse auf irgendwelche anderen Merkmalträger zu.

Metamodellebene

Eine fundamentale Annahme des Merkmalmodells ist, dass es eine Metamodellebene gibt, durch die eine Vergleichbarkeit zwischen mehreren merkmalbasierten Modellen erst hergestellt wird. Diese Metamodelle existieren für Merkmalträger, Merkmale und Ausprägungsaussagen. Im Fall von Merkmalträgern wird das Metamodell als „Merkmalträgertyp“ bezeichnet. Der Merkmalträgertyp definiert die Merkmale, die ein Merkmalträger dieses Typs besitzt. Sind beispielsweise zwei Produkte Merkmalträger vom selben Merkmalträgertyp, dann ist damit klar, dass an beiden Produkten gleiche Merkmale messbar sind, was eine Grundvoraussetzung für den Vergleich der Produkte ist. Eine weitere Voraussetzung ist, dass bei der Messung der Ausprägung dasselbe Messverfahren angewendet wird. Dies wird dadurch abgesichert, dass Merkmale ebenfalls jeweils einen Typ, genannt „allgemeines Merkmal“, besitzen. Das allgemeine Merkmal hat eine im Merkmalträgertyp eindeutige Bezeichnung und definiert das Messverfahren, das zur Bestimmung der Ausprägung angewendet wird. Der Merkmalträgertyp verweist also auf eine Liste von allgemeinen Merkmalen, wodurch klar ist, welche Merkmale an einem Merkmalträger vorhanden sind und wie ihre Ausprägung bestimmt wird. Ähnlich verhält es sich mit Ausprägungsaussagen. Eine Ausprägungsaussage bezieht sich stets auf eine Aussageart, die die Semantik der einzelnen Aussage definiert. Auch hier ist der Vergleich zweier Aussagen nur möglich, wenn die Semantik beider Aussagen gleich ist oder zumindest deren Beziehung klar definiert ist. Beispielsweise kann eine Aussage eine Anforderung an eine Ausprägung sein und eine andere Aussage eine Zusicherung für dasselbe Merkmal. Ob die Anforderung durch die Zusicherung erfüllt wird, kann aber nur entschieden werden, wenn die Anforderung die Art Zusicherung auch explizit akzeptiert. Zum Beispiel kann die Einbeziehung von Garantieleistungen in eine Zusicherung ein wichtiges Kriterium für die Erfüllung einer Anforderung sein.

Es ist auch auf der Ebene von Merkmalträgertypen möglich, Ausprägungsaussagen zu treffen. Diese beziehen sich dann nicht auf ein konkretes Merkmal eines Merkmal-

trägers, sondern betreffen grundsätzlich alle Merkmalträger dieses Merkmalträgertyps. Beispielsweise werden in Katalogen Produkte immer auf der Typ-Ebene beschrieben, trotzdem werden dort konkrete Angaben über die Merkmale der einzelnen Produkte gemacht, die diesem Typ entsprechen. Der Herausgeber des Katalogs sichert in dem Fall zu, dass jeder einzelne Merkmalträger die im Katalog genannten Eigenschaften hat.

Die bis hierhin beschriebenen Grundbegriffe werden in Abbildung 2.1 dargestellt. Eine explizite Modellierung von Merkmalen auf der Modellebene wurde darin ausgelassen, weil es implizit durch Merkmalträgertyp (Existenz) und Ausprägungsaussage (zugeordnete Werte) beschrieben wird.

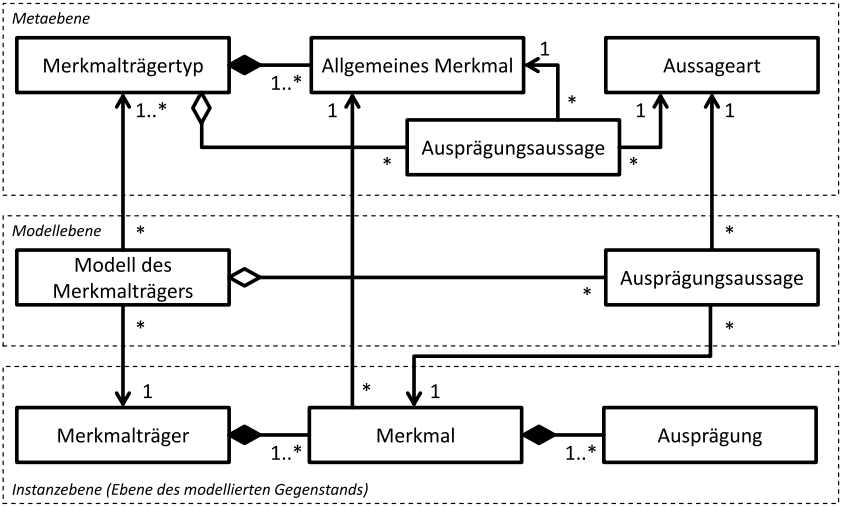

Abbildung 2.1.: Beziehung der Grundbegriffe des Merkmalmodells, dargestellt in der Notation eines UML-Klassendiagramms.

Merkmalart und Merkmalprototyp

Grundsätztlich sind die bisher eingeführten Grundbegriffe ausreichend, um Merkmalträger zu modellieren. Allerdings müsste dann für jeden Merkmalträgertyp jedes allgemeine Merkmal einzeln vollständig definiert werden. Zum Beispiel haben viele Merkmalträger „Höhe“ und „Breite“ als Merkmale und es wäre unsinnig, für jeden Merkmalträgertyp diese allgemeinen Merkmale erneut zu definieren. Daher beziehen sich allgemeine Merkmale auf Merkmalarten. Eine Merkmalart hat, anders als das allgemeine Merkmal, keinen Bezug zu einem bestimmten Merkmalträgertyp und beinhaltet daher auch keine genaue Vorschrift zur konkreten Messung der Ausprägung. Trotzdem können durch eine Merkmalart bereits viele allgemeingültige Aussagen getroffen wer-

den, die dann für jedes zugeordnete allgemeine Merkmal gelten. Beispielsweise können Randbedingungen für Messverfahren vorgegeben werden.

Merkmalarten beziehen sich wiederum auf genau einen Merkmalprototyp. Der Merkmalprototyp hat überhaupt keinen Sachbezug, sondern legt nur fest, welche Art von Größe eine Merkmalart ist. Die Merkmalarten „Höhe“ und „Breite“ haben also den gemeinsamen Merkmalprototyp „Länge“ im Sinne einer räumlichen Distanz.

Verebungsmechanismen

Vererbung im Sinne der Objektorientierung bezeichnet, allgemein ausgedrückt, die Bildung einer hierarchischen Ordnung zwischen Objekten, in der sich die Eigenschaften eines hierarchisch übergeordneten Objekts in allen direkt untergeordneten Objekten wiederfinden, das heißt an sie „verebt“ werden. Vererbte Eigenschaften werden ebenfalls weitervererbt. Vererbung wird im Merkmalmodell für Merkmalarten und Merkmalträgertypen angewendet. Prinzipiell lässt er sich auch für Aussagearten verwenden (wie in [54] beschrieben), zur Vereinfachung des Modells wird hier aber darauf verzichtet.

In beiden genannten Fällen werden sämtliche Informationen, die über den abstrakteren (das heißt in der Vererbungshierarchie höheren) Merkmalträgertyp beziehungsweise Merkmalart vorhanden sind, vererbt. Konkret sind das die allgemeinen Merkmale eines Merkmalträgertyps und die getroffenen Aussagen über eine Merkmalart. Im Vergleich zur Vererbung, wie sie in objektorientieren Programmiersprachen verwendet wird, gibt es jedoch einen wichtigen Unterschied: Viele Programmiersprachen erlauben das sogenannte „Überschreiben“ von vererbten Eigenschaften, so dass beispielsweise eine vererbte Funktion unter derselben Bezeichnung inhaltlich anders implementiert wird. Dieser Mechanismus des Überschreibens ist im Merkmalmodell nicht vorhanden. Vererbte Information steht unveränderlich fest.

Die Vererbungshierarchie für Merkmalträgertypen ist nicht immer eine Monohierarchie. Viele Merkmalträger vereinen Merkmale aus mehreren Merkmalträgertypen, was sich auf der Metamodellebene durch mehrfaches Erben eines Merkmalträgertyps von anderen Merkmalträgertypen ausdrücken lässt. Beispielsweise kann ein Gerätetyp mehrere andere Gerätetypen integrieren und deshalb auch deren allgemeine Merkmale erben. Weil jedes der vererbten allgemeinen Merkmale eine eigene, unabhängige und unveränderliche Definition besitzt, führt die Mehrfachvererbung für Merkmalträgertypen zu keinerlei Konflikten. Abbildung 2.2 zeigt die Metaebene des Merkmalmodells mit Merkmalart, Merkmalprototyp und Vererbungsbeziehungen.

2.1.2. Dienstbasierte Verwendung von Merkmalen

Eine wichtige Verwendung des Merkmalmodells liegt im Informationsaustausch zwischen technischen Systemen. Daher ist neben dem Merkmalmodell auch die Kommunikation über Merkmale ein Aspekt, zu dem es Vereinbarungen über Systemgrenzen hinweg geben muss. Ein wichtiges Grundkonzept für den Informationsaustausch zwischen vernetzten Systemen ist der Dienst. In der Automatisierungstechnik werden Dienste als wichtige zukünftige Basistechnologie angesehen (s. z.B. [25]).

Das Kernmodell von Diensten ist in Abbildung 2.3 vereinfacht dargestellt. Es ist der DIN SPEC 40912 [24] entnommen.

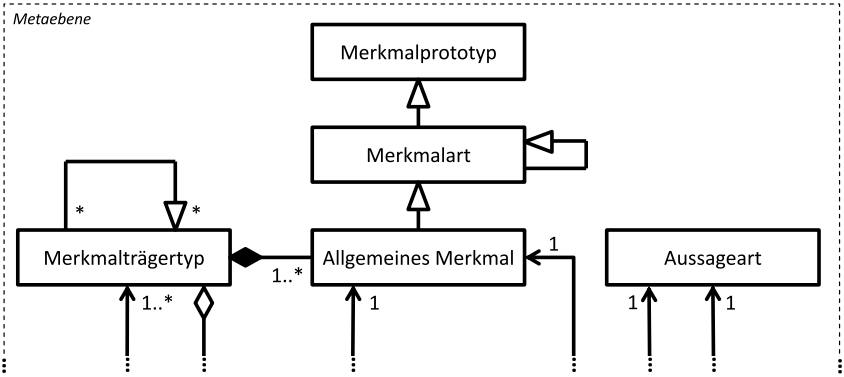

Abbildung 2.2.: Um Vererbungsbeziehungen erweiterte Metaebene des Merkmalmodells, dargestellt in der Notation eines UML-Klassendiagramms.

Grundsätzlich ist ein Dienst ein Organisationsschema durch das Leistungsbedarfe und Leistungen (i.S.v. Dienstleistungen) zusammengeführt werden (Definition entsprechend [60]). Die Leistung wird von einem Dienstleister auf Anforderung eines Benutzers hin erbracht. Der eigentliche Dienst ist eine Sicht des Benutzers auf die Art der Leistungserbringung und daher ein gedankliches Konstrukt ohne eine bestimmte physische Repräsentation.

Dem Benutzer von Diensten sind beliebige Diensttypen bekannt. Durch den Diensttyp kennt der Benutzer die Semantik eines konkreten Dienstes. Die Verwendung von Diensten geschieht durch diskrete Dienstaufrufe und Ergebnisse der Aufrufe werden gegebenenfalls als diskrete Ereignisse (wie zum Beispiel Nachrichten) an den Benutzer zurückgegeben. Zu jedem Dienst gehört eine Menge von Operationen, die beim Dienstaufruf angegeben werden. Die Ausführung dieser Operationen ist Aufgabe des Dienstleisters, der für den Benutzer gegebenenfalls verborgen ist. Die Operationen müssen jedoch folgende Eigenschaften aufweisen:

- Sie sind einzeln aufrufbar.
- Ihre Funktionalität ist jeweils einzeln definiert.
- Sie gehören exklusiv zu einem Dienst.
- Sie werden atomar ausgeführt.
- Sie arbeiten alle mit derselben Menge von Objekten.
- Dienstaufrufe eines Benutzers werden in der Reihenfolge des Eintreffens bearbeitet.

Zusätzlich zum allgemein festgelegten Diensttyp kennt der Benutzer Zusicherungen über die Qualitätsmerkmale zu jedem ihm bekannten Dienst. Der Dienstleister garantiert die Einhaltung dieser Zusicherungen.

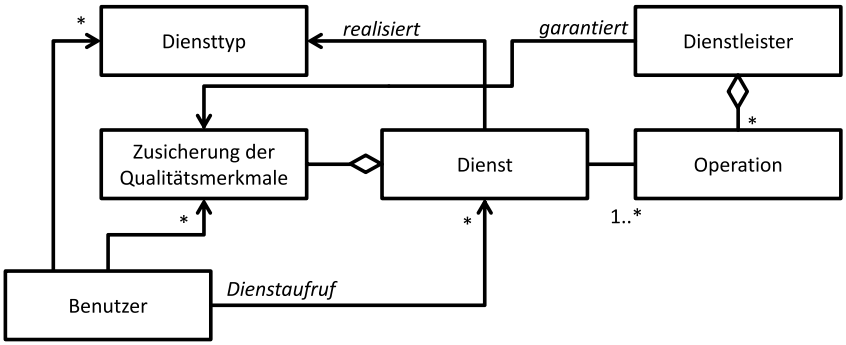

Abbildung 2.3.: Kernmodell „Dienst“ als UML-Klassendiagramm (vereinfacht).

Wesentliches Kennzeichen eines Dienstes ist, dass Dienstleister und Benutzer durch den Dienst voneinander getrennt sind. Dadurch werden zwei wichtige Eigenschaften erzeugt: Die Kapselung des Dienstleisters und die lose Kopplung zwischen Benutzer und Dienstleister. Für die Verwendung zum Zugriff auf technische Merkmale sind diese Eigenschaften deshalb wichtig, weil die Lebenszyklen von Benutzer und Dienstleister vollkommen unabhängig sind. Sowohl ein Automatisierungssystem als Benutzer als auch ein Softwaresystem als Dienstleister können über Jahre, eventuell sogar über Jahrzehnte hinweg betrieben werden und innerhalb dieser Zeit mit vielen unterschiedlichen Dienstleistern beziehungsweise Benutzern interagieren. Die lose Kopplung aneinander und die Kapselung der inneren Implementierung sind daher wichtige Voraussetzungen für einen unabhängigen und wartungsarmen Betrieb.

Bezogen auf die Kommunikation technischer Merkmale, und unter Anwendung des in Abschnitt 2.1.1 eingeführten Merkmalmodells, soll dieses Dienstmodell nun konkretisiert werden. Der Dienstleister ist in diesem Fall ein System, das den Zugriff auf Merkmalinformation erlaubt. Benutzer sind technische Systeme oder Menschen, die diese Information lesen, schreiben oder verändern möchten. Damit sind der Diensttyp und die notwendigen Operationen grundsätzlich festgelegt: Der Diensttyp muss in jedem Fall das Merkmalmodell referenzieren, damit die Begriffe und Begriffsbeziehungen für die Dienstverwendung festgelegt sind. Mit diesen Begriffen können dann die ausgetauschten Informationen benannt werden. Die Operationen, die der Dienstleister implementiert, betreffen Daten entsprechend des Merkmalmodells. Sie können aber je nach Anwendungsfall und Fähigkeiten des Dienstleisters mehr oder weniger umfangreich ausfallen. Dies wird im folgenden Abschnitt genauer behandelt.

Die Qualitätsmerkmale eines Dienstes spielen in dieser Betrachtung eine untergeordnete Rolle. Ihr Umfang und Inhalt hängt nicht nur vom Dienstleister ab, sondern auch von den Anforderungen, die Benutzer stellen (z.B. Verfügbarkeit und Reaktionszeit). Annahmen darüber sollen an dieser Stelle jedoch nicht vorweggenommen werden.

Klassifikation von Diensten für Merkmalsysteme

Das Merkmalmodell beschreibt – wie in diesem Kapitel eingangs erwähnt – ein systematisches Vorgehen bei der Modellierung auf der Modell- und Metamodellebene. Die allgemeine Dienstleistung, die damit verknüpft werden kann, ist daher das Erstellen, Verändern und Auslesen von Modellen, die mit dem Merkmalmodell konform sind. Ein technisches System, wie beispielsweise ein Server, das diese Dienstleistung implementiert, wird im Folgenden als „Merkmalsystem“ bezeichnet. In der Softwaretechnik kommen die genannten Grundoperationen für die Verwaltung von Daten häufig vor. Sie werden oft abkürzend mit den Buchstaben c (create), r (read), u (update) und d (delete) bezeichnet. Die Operation zum Lesen kann dabei spezifisch für ein Objekt ausgeführt werden oder alle Objekte einer hierarchischen Ebene, zum Beispiel alle Aussagen über ein Merkmal, zurückgeben. Grundsätzlich kann ein Dienstleister alle diese Operationen für jedes Objekt in einem Merkmalmodell anbieten und wird, sofern die Objektbeziehungen berücksichtigt werden, in sich konsistente Merkmalmodelle zusichern können. Ziel in der Verwendung des Merkmalmodells ist aber nicht nur die interne Konsistenz, sondern auch Konsistenz mit anderen merkmalverwaltenden Systemen. Wird beispielsweise ein Produkt in einer Datenbank durch Aussagen über ein Merkmal x charakterisiert, dann muss zur Vergleichbarkeit mit anderen Produkten, die ebenfalls über das Merkmal x verfügen, dieses Merkmal als allgemeines Merkmal an einer dritten (unabhängigen) Stelle definiert sein. Selbstverständlich darf sich diese Definition auch nicht ändern und muss referenzierbar sein. Für die systemübergreifende Konsistenz von Modellen sind einige Operationen, wie das Verändern oder Löschen eines allgemeinen Merkmals, daher problematisch. Allgemein ist die Frage, welche Operationen in welchen Arten von merkmalverwaltenden Systemen verfügbar sein müssen beziehungsweise dürfen, eine Frage des jeweiligen Zwecks des Systems. Für den Benutzer bildet das ein sinnvolles Kriterium zur Klassifikation von Dienstleistern, beispielsweise solchen, die allgemeine Merkmale definieren, solchen, die Merkmalträgertypen verwalten und solchen, die Informationen über konkrete Merkmalträger anbieten.

Eine dementsprechende Klassifikation von Dienstleistern wurde in [48] und [49] vorgestellt. Es werden fünf Gruppen von Dienstleistern (M1, ..., M5) unterschieden:

- M1: Nur Lesezugriff
  - Lesen aller Modelle und Modellinhalte
  - Navigation zwischen Modellen
- M2: Operativer Betrieb in der Automatisierungstechnik
  - Wie M1, zusätzlich Aktualisierung von Aussagewerten
- M3: Modellierung von Merkmalträgern
  - Wie M2, zusätzlich:
  - Erstellen, Verändern und Löschen von Merkmalträgern
  - Erstellen, Verändern und Löschen von Ausprägungsaussagen
- M4: Modellierung von Merkmalträgertypen
  - Erstellen von Merkmalträgertypen

- M5: Normung und Standardisierung
  - Erstellen von Merkmalarten, Aussagearten und Merkmalprototypen

Die verfügbaren Operationen werden in Tabelle 2.1 aufgelistet. Für den operativen Betrieb von Automatisierungssystemen sind solche Merkmalsysteme von Interesse, die zu den Gruppen M1 und M2 gehören. Beispielsweise können in einer Produktionsanlage die Produkt- oder Gerätedaten während der laufenden Produktion ausgelesen werden, oder es können solche Daten, die erst während der Produktion bekannt werden, in das System eingetragen werden, zum Beispiel zur Dokumentation des Produktlebenszyklus'. Die übrigen Gruppen ermöglichen Vorgänge zur Modellierung und Metamodellierung, die im operativen Betrieb eines Automatisierungssystems bereits abgeschlossen sind.

Tabelle 2.1.: Je Zelle sind die verfügbaren Operationen für den Zugriff auf Merkmalmodelle aufgelistet. M1 bis M5 bezieht sich auf die Liste auf Seite 17. c: Create/Erstellen, r: Read/Lesen, u: Update/Verändern, d: Delete/Löschen.

Modellelement \ Gruppe	M1	M2	M3	M4	M5
Merkmalprototyp	-/r/-/-	-/r/-/-	-/r/-/-	-/r/-/-	c/r/-/-
Merkmalart	-/r/-/-	-/r/-/-	-/r/-/-	-/r/-/-	c/r/-/-
Aussageart	-/r/-/-	-/r/-/-	-/r/-/-	-/r/-/-	c/r/-/-
Merkmalträgertyp	-/r/-/-	-/r/-/-	-/r/-/-	c/r/-/-	-/-/-/-
Merkmalträger	-/r/-/-	-/r/-/-	c/r/u/d	c/r/u/d	-/-/-/-
Allgemeines Merkmal	-/r/-/-	-/r/-/-	-/r/u/-	c/r/u/-	-/-/-/-
Ausprägungsaussage	-/r/-/-	-/r/-/-	c/r/u/d	c/r/u/d	-/-/-/-
Aussagewert	-/r/-/-	-/r/u/-	c/r/u/-	c/r/u/-	-/-/-/-

Höherwertige Dienste

Die aufgeführten Dienste ermöglichen gemeinsam den vollständigen Zugriff zum Lesen und Manipulieren der Information in einem Merkmalsystem. Ob diese Möglichkeiten auch in praktischer Hinsicht ausreichen, hängt vom Anwendungsfall ab. Nehmen wir beispielsweise an, dass ein Benutzer Merkmalträger mit bestimmten Eigenschaften innerhalb eines Merkmalsystems sucht – beispielsweise ein Produkt, für das gewisse Mindestanforderungen an bestimmte Merkmale gelten. Der Benutzer kann nun alle im Merkmalsystem hinterlegten Merkmalträger abrufen und auf Übereinstimmung mit den Anforderungen prüfen. Diese Vorgehensweise ist aber aus mehreren Gründen nachteilig. Erstens tritt dieser Anwendungsfall vermutlich bei vielen Benutzern auf und es ist daher effizienter die Implementierung des entsprechenden Algorithmus einmalig im Merkmalsystem zu hinterlegen. Zweites ist die Kommunikationslast für Merkmalsystem, Benutzer und Kommunikationsmedium höher, wenn die Suche klientenseitig erfolgt. Drittens skaliert der Aufwand zur Ausführung des Algorithmus mit der Menge zu durchsuchender Daten. Informationsverarbeitende Systeme, die als Benutzer auftreten, können mit diesem Aufwand überfordert sein, weil ihre Rechenkapazität in keinem Zusammenhang mit der Informationsmenge im benutzten Merkmalsystem steht – im

Gegensatz dazu kann das Merkmalsystem mit der Datenmenge skaliert werden. Letztendlich ist die Implementierung höherwertiger Dienste auf Seite des Merkmalsystems in den meisten Fällen die bessere Alternative, so dass neben den genannten Basisdiensten auch höherwertige Dienste vom Merkmalsystem angeboten werden sollten. In Kapitel 5 werden solche Dienste definiert.

2.2. Informationssysteme

In diesem Abschnitt werden Grundlagen der Theorie von Informationssystemen erläutert, genauer gesagt die Theorie relationaler Datenbanken. Diese Theorie ist eine umfangreiche, mächtige, verbreitete und anerkannte formale für Datenbanken und Abfragesprachen. Es gibt drei Gründe, diese formale Grundlage innerhalb dieser Arbeit zu betrachten: Erstens schafft die formale Darstellung Klarheit. Zweitens lassen sich Erkenntnisse aus der vorhandenen Theorie übertragen. Drittens basieren viele vorhandene Informationssysteme an dieser Theorie, so dass eine Integration des in dieser Arbeit erstellten Konzepts vereinfacht wird. Es ist jedoch kein erklärtes Ziel dieser Arbeit, dass eine technische Umsetzung des Konzepts speziell für relationale Datenbanken angestrebt wird.

Im folgenden Abschnitt werden die Grundbegriffe des relationalen Datenmodells eingeführt. Die Inhalte beschränken sich dabei auf die innerhalb dieser Arbeit relevanten Aspekte. Für eine vollständige Einführung wird auf entsprechende Fachliteratur wie beispielsweise [11] verwiesen.

2.2.1. Grundbegriffe des relationalen Datenbankmodells

Das relationale Datenbankmodell wurde bereits 1970 von Codd eingeführt [9]. Grundlegendes Element ist die Relation. Eine Relation ist eine Tabelle, in der zeilenweise Datensätze eingetragen sind, das heißt je Zeile ein Datensatz. Eine Zeile wird dabei in der Terminologie relationaler Datenbanken Tupel genannt. Die Spalten der Tabelle werden als Attribute bezeichnet und beinhalten je Tupel entweder einen oder keinen Wert. Jedes Attribut einer Relation ist eindeutig benannt. Damit Tupel innerhalb einer Datenbank eindeutig identifiziert werden können, hat jede Relation einen eindeutigen Namen und alle Tupel einer Relation sind paarweise mindestens durch den Wert eines bestimmten Attributs (oder einer Attributkombination) verschieden. Dadurch kann in jeder Relation ein sogenannter Primärschlüssel bestimmt werden, das heißt ein oder mehrere Attribute, durch deren Werte jedes Tupel eindeutig bestimmt werden kann.

Die Werte von Attributen können im allgemeinen Fall eine Zahl, eine Zeichenkette oder ein boolescher Wahrheitswert sein, jedoch wird der mögliche Wertebereich in vielen Anwendungen entsprechend der Semantik des Attributs eingeschränkt, beispielsweise so, dass für das Attribut „Datum“ auch tatsächlich nur ein Datum als Wert eingetragen werden kann.

Zur einheitlichen Darstellung von Relationen werden sogenannte Relationsschemas verwendet. Diese beinhalten eine Menge von Attributnamen und die jeweils gültigen Wertebereiche. Beispielweise kann

$$BESTELLUNG = \left\{ Nummer : \mathbb{N}, Kundennr : \mathbb{N}, Produkt : \Sigma^*, Menge : \mathbb{R}^+ \right\}$$

als Relationsschema für eine Datenbank mit Bestellungen verwendet werden (Σ^* sei hier die Menge aller Zeichenketten). Für umfangreichere Datenbanken werden grafische Sprachen wie Entity-Relationship-Diagramme [8] zur Definition von Relationsschemas verwendet. In dieser Arbeit wird das aber nicht notwendig sein.

2.2.2. Relationale Algebra

Grundlage von heute üblichen Abfragesprachen wie z.B. SQL ist die relationale Algebra, die an dieser Stelle nur knapp und informell eingeführt werden soll. Für eine präzisere und formale Definition wird auf entsprechende Fachliteratur wie [72] verwiesen.

Die relationale Algebra ist eine formale Sprache, die durch Operatoren Elemente des relationalen Datenbankmodells, d.h. Mengen von Tupeln, miteinander verknüpft. Die Ergebnisse solcher Operationen sind erneut Tupelmengen. Die Liste von Operatoren der relationalen Algebra ist nicht endlich, auch wenn Codd ursprünglich acht Operationen definiert hatte [10]. Je nach Zielsetzung können zusätzliche Operationen definiert werden. Relationale Datenbanken unterstützen jedoch in den meisten Fällen nur Grundoperationen und zusätzliche spezielle Operationen abseits der Algebra.

Die relationale Algebra ist für Informationssysteme in theoretischer und praktischer Hinsicht von großer Bedeutung. Durch die zugrunde liegenden Formalismen lassen sich weitreichende Aussagen über das relationale Datenmodell treffen, beispielsweise über die Berechenbarkeit von Datenabfragen. Aus praktischer Sicht bilden die Operationen der relationalen Algebra eine Grundmenge von Funktionen, aus der eine Abfragesprache gebildet werden kann. Die folgenden Abschnitte stellen einige typische praktisch vorkommende Operationen informell vor. Dies ist keine Beschreibung der vollständigen ursprünglichen relationalen Algebra, sondern eine praktisch motivierte Auswahl.

Auswahl Die Operation „Auswahl“ gibt nur die Tupel einer Relation R zurück, für die ein Prädikat P erfüllt ist. Das übliche Formelzeichen für Auswahl ist σ. Somit ist

$$\sigma_P(R)$$

die Schreibweise für die Auswahl-Operation. Das Prädikat P ist ein Boolescher Ausdruck der sich auf ein oder mehrere Attributwerte der Tupel in R bezieht. Beispielsweise würde

$$\sigma_{BESTELLUNG.Produkt=\mathrm{XY23}}(Bestellungen)$$

alle Bestellungen des Produkts XY23 zurückgeben (*Bestellungen* entspricht dem Schema $BESTELLUNG$).

Projektion Die Projektion reduziert eine Relation R auf bestimmte Attribute $a_1, ..., a_n$ und gibt eine Relation zurück, die nur $a_1, ..., a_n$ enthält. Die Schreibweise ist

$$\pi a_1, ...a_n(R).$$

Soll beispielsweise in der oben genannten Auswahl aller Bestellungen des Produkts XY23 nur die Kundennummer ausgegeben werden, dann kann das durch den Ausdruck

$$\pi_{Kundennummer}(\sigma_{BESTELLUNG.Produkt=\mathrm{XY23}}(Bestellungen))$$

geschehen.

Kartesisches Produkt und Vereinigung Das kartesische Produkt entspricht der gleichnamigen Operation der Mengenlehre. Das heißt aus zwei Relationen R_1 und R_2 wird eine neue Relation erstellt, in der jede mögliche Kombination von Tupeln aus R_1 und R_2 jeweils ein neues Tupel bildet. Dies wird durch

$$R_1 \times R_2$$

notiert.

Die Vereinigung ist eine Operation, die als Folge von kartesischem Produkt und Auswahl ausgedrückt werden kann. Sie kann daher als

$$R_1 \bowtie_P R_2 := \sigma_P(R_1 \times R_2)$$

definiert werden, wobei $\bowtie$ das Symbol für die Vereinigungsoperation ist und P ein zu erfüllendes Prädikat. Das Prädikat entscheidet darüber, welche Paare von Tupeln gebildet werden. Die Operationen $\bowtie$ und $\times$ sind für $P \equiv true$ identisch. In der Praxis wird $\bowtie$ oft zum Zusammenführen von zwei Relationen verwendet. Nehmen wir beispielsweise an, dass ein Relationsschema $KUNDE$ mit den Attributen Name und Kundennummer existiert, und dass $Kunden$ dem Schema $KUNDE$ entspricht, dann kann durch

$$Kunden \bowtie_{KUNDE.Kundennr.=BESTELLUNG.Kundennr.} Bestellungen$$

eine Relation mit allen Bestellungen inklusive Kundennamen gebildet werden.

Differenz Die Differenz $R_1 \backslash R_2$ zweier Relation R_1 und R_2 enthält genau die Tupel aus R_1, die nicht in R_2 vorhanden sind. Somit kann die Operation zum Entfernen von Duplikaten aus zwei Relationen verwendet werden. Voraussetzung ist, dass die Relationen demselben Schema entsprechen.

Umbenennung Durch die Umbenennung $\rho_{b/a}(R)$ wird eine Relation erzeugt, in der das Attribut a in b umbenannt ist. Somit lassen sich Mengenoperationen wie die Differenzbildung mithilfe von Umbenennungen auch dann durchführen, wenn die involvierten Relationen nicht demselben Schema entsprechen.

2.2.3. Relationenkalkül

Der Relationenkalkül ist eine Form der Formulierung von Abfragen an relationale Datenbanken, bei der das Ergebnis der Abfrage durch freie Variablen deklariert wird. Dazu werden Mengenoperationen und Prädikatenlogik verwendet. Anders als bei der relationalen Algebra bestehen Ausdrücke des Relationenkalküls also nicht aus einer Vorschrift zur Berechnung des Ergebnisses, sondern es werden nur die Eigenschaften des Ergebnisses definiert.

Es gibt zwei wesentliche Formen des Relationenkalküls: Den sogenannten Tupel-Kalkül [10] und den Wertebereichs-Kalkül [50]. In beiden Fällen wird eine Anfrage als Menge von Variablenbelegungen angegeben, für die ein Prädikat erfüllt ist. Der Unterschied ist, dass diese Variablen im Tupel-Kalkül Tupel als Werte besitzen, während sie

im Wertebereich-Kalkül Attributwerten entsprechen. Soll beispielsweise abgefragt werden, für welche Kunden eine Bestellung des Produkts XY23 vorliegt, dann würde das im Tupel-Kalkül durch den Ausdruck

$$\{k|\exists b : (b.Kundennr = k.Kundennr \wedge b.Produkt = \text{XY23})\}, \\ k \in Kunden, b \in Bestellungen,$$

formuliert werden. Im Wertebereich-Kalkül ließen sich stattdessen Name und Kundennummer als Werte (nicht als ein Tupel) durch den Ausdruck

$$\{n, kNr|\exists kNr \exists p : (Kunden(n, kNr)) \wedge (Bestellungen(bNr, kNr, p, m) \wedge p = \text{XY23})\}$$

deklarieren (die Wertebereiche der Variablen sind mit denen der jeweiligen Attribute identisch).

2.2.4. Eigenschaften der relationalen Algebra

Aufgrund der formalen Basis ist es möglich, grundsätzliche theoretische Eigenschaften relationaler Datenbanken zu zeigen. Dadurch ergeben sich weitreichende Konsequenzen für den Entwurf und die Verwendung von Informationssystemen und Abfragesprachen.

Die Ergebnisse von Operationen der relationalen Algebra sind ebenfalls Relationen und damit selbst wieder von Operationen verwendbar. Sie besitzt daher die theoretische Eigenschaft der Abgeschlossenheit. In der praktischen Anwendung können Operationen daher beliebig geschachtelt werden, ohne dass dabei Restriktionen beachtet werden müssen. Das trägt zu einer einfachen Anwendung bei und erhöht die Verwendungsmöglichkeiten.

Es lässt sich formal zeigen, dass bestimmte Mengen von Operationen der relationalen Algebra „relational vollständig“ sind [10]. Das bedeutet, dass sich durch die Operationen einer relational vollständigen Menge und durch deren Verknüpfung jeder Ausdruck der relationalen Algebra nachbilden lässt. Die oben beschriebenen Operationen sind ein Beispiel für eine relational vollständige Menge von Operationen. Die Eigenschaft der relationalen Vollständigkeit dient als ein Indikator für die Ausdruckstärke einer Abfragesprache. Ist die Sprache relational vollständig, dann ist die Ausdruckstärke mindestens äquivalent zur relationalen Algebra.

Andererseits ist durch die relationale Vollständigkeit nicht garantiert, dass sich jede Abfrage, die möglicherweise von Interesse wäre, auch formulieren lässt. Ein bekanntes Beispiel dafür ist das Fehlen einer transitiven Hülle. Wenn zum Beispiel ein Lebenszyklus in einer Relation so dokumentiert wird, dass jedes Tupel einem einzelnen Teilprozess des Lebenszyklus entspricht und jeweils auf den nachfolgenden Teilprozess verweist, dann kann durch relationale Algebra keine Abfrage formuliert werden, die alle Teilprozesse der Reihe nach auflistet. In einem solchen Fall müsste eine Datenbank zusätzliche Operationen anbieten, die nicht mit Mitteln der relationalen Algebra ausgedrückt werden können.

Eine weitere praktisch wichtige Einschränkung ist das Fehlen von aggregierenden Operationen. Zum Beispiel kann mit Mitteln der relationalen Algebra zwar festgestellt werden, welche Tupel einer Relation eine Bedingung erfüllen, aber nicht wie viele. Das Aufsummieren von Werten mehrerer Tupel ist ebenfalls nicht möglich. Das ist bei Datenabfragen in der Praxis eine recht starke Einschränkung, wenn man bedenkt, dass solche

Aufgaben in vielen Fällen anfallen und dass die einzige Alternative ist, alle Tupel abzufragen und dann die Aggregation beim Abfragenden selbst durchzuführen. Dieses Vorgehen belastet die Kommunikation und den Klienten. Aus diesem Grund bieten Abfragesprachen wie SQL üblicherweise auch Aggregationsoperationen an und beschränken sich nicht auf die reine relationale Algebra.

Eine wichtige, formal beweisbare Eigenschaft ist, dass relationale Algebra und die unterschiedlichen Formen des Relationenkalküls gleich mächtig sind [11]. Das bedeutet, dass es für jede Formel in relationaler Algebra einen entsprechenden Ausdruck des Relationenkalküls gibt und umgekehrt. Je nach Anwendungsgebiet kann also die eine oder die andere Darstellungsform verwendet werden, ohne dass dadurch Einschränkungen entstehen.

3. Stand der Technik

3.1. Merkmal-Modelle in der Praxis

Im Umfeld industrieller Automation sind heute zahlreiche Normen und Standards vorhanden, die eine einheitliche, maschinell lesbare Verwendung von Merkmalen zum Ziel haben. Der Hauptanwendungsbereich liegt in der elektronisch abgewickelten Beschaffung, beispielsweise in der automatisierten Abwicklung von Bestellprozessen für Geräte. Die Normen und Standards behandeln meist die Meta-Metaebene, das heißt die grundsätzliche Art der Abbildung von Merkmalinformation, während die Metaebene, das heißt die Festlegung von allgemeinen Merkmalen von Merkmalträgertypen, in Klassifikations- und Katalogsystemen behandelt wird. Die Abbildung einzelner Merkmalträger wird weit weniger häufig adressiert. Heeg [32] und Mertens [54] haben in ihren Arbeiten diverse Quellen ausführlich behandelt und zählen insgesamt 18 Normen und Standards, die das Thema „Merkmale“ direkt adressieren, die jedoch auch zum großen Teil aufeinander aufbauen bzw. zwischenzeitlich harmonisiert wurden. Dadurch hat sich die IEC 61360 [37], die mit weiteren Normenreihen wie ISO 10303 [42], ISO 13584 [43] und DIN 4002 [23] abgestimmt ist, als wichtige Grundlagennorm der Meta-Metaebene etabliert.

Im Bereich der Klassifikations- und Katalogsysteme zeichnet sich, zumindest für die Industrie im deutschsprachigen Raum, eine ähnliche Konsolidierung ab. Weltweit wurden mehrere Klassifikationssysteme entwickelt und mit Listen von (allgemeinen) Merkmalen und Merkmalträgertypen gefüllt. Besonders bedeutende und große Klassifikations- und Katalogsysteme sind UNSPSC, eCl@ss und RosettaNet. Diese Systeme haben alle das erklärte Ziel, den elektronischen Austausch von Produktdaten durch standardisierte Merkmale zu erleichtern. Sie wurden aber jeweils von unterschiedlichen Interessengruppen ins Leben gerufen und sind daher in unterschiedlichen Anwendungsgebieten gebräuchlich. In einer Studie aus dem Jahr 2005 wird das auch in Hinblick auf die Inhalte der Katalogsysteme bestätigt, denn alle dort betrachteten Systeme (UNSPSC, eCl@ss, RosettaNet und eOTD) sind in Bezug auf die Klassen von standardisierten Merkmalträgertypen sehr unausgeglichen besetzt [33]. Welcher Standard für die Industrie in Deutschland zum aktuellen Zeitpunkt den höchsten Stellenwert hat, lässt sich durch belastbare Fakten schwer direkt belegen. Allerdings bescheinigt eine Studie aus dem Jahr 2002 eCl@ass mit 32.4% den größten Anteil unter den genutzten Klassifikationsstandards in der Elektroindustrie und -großhandel [61]. Am zweihäufigsten wurde der ETIM-Standard genutzt (18.9%). Seitdem wurde eCl@ss mit ETIM sowie proficl@ss und Prolist harmonisiert und soll mit der IEC 61987 [38] harmonisiert werden [31], so dass eCl@ss vermutlich noch immer vorherrschend und mit anderen bedeutenden Standards kompatibel ist. Das Datenmodell von eCl@ass basiert auf der IEC 61360. Für die Abbildung in Dateien sind jedoch unterschiedliche Dateiformate im Gebrauch. Gerätehersteller bieten beispielsweise oft über ihre Webseiten eCl@ss-konforme Kataloge im

Format „BMEcat“ an.

Ein weiteres wichtiges Anwendungsgebiet von technischen Merkmalen ist das Engineering automatisierter Systeme. Hier werden Merkmale insbesondere für die Beschreibung von einzelnen Geräten, beispielsweise Sensoren und Aktoren, verwendet, um deren Konfiguration zu vereinfachen. Besonders bedeutend ist die „Electronic Device Description Language“ (EDDL) [41], durch die standardisierte Gerätebeschreibungen ermöglicht werden. Diese Beschreibungssprache umfasst neben Merkmalen noch viele weitere Aspekte von Geräten. Die Modellierung von Merkmalen selbst steht dabei im Hintergrund und fällt daher verhältnismäßig einfach aus. Von wissenschaftlicher Seite wurde das Potenzial einer konsequenten Verwendung von standardisierten Merkmalen für Anwendungen im Engineering identifiziert (s. z. B. [63, 67]), die Abläufe sind hier jedoch komplex und Anwendungsmöglichkeiten zahlreich. Die Konsolidierung und Zusammenführung von Modellen ist daher weniger fortgeschritten als im Anwendungsbereich e-Commerce.

Für die praktische Anwendung von Merkmalmodellen ist die Ausgangslage insgesamt gut: Es gibt heute auf der Metameta- und auch auf der Metaebene anerkannte Normen bzw. Standards im allgemeineren Sinne, die als Grundlage zur Modellierung von Merkmalen dienen und umfangreiche Kataloge von Merkmalträgertypen und allgemeinen Merkmalen zur Verfügung stellen (eCl@ass enthält nach eigenen Angaben aktuell 40800 Merkmalträgertypen [26]). Trotz dieser Situation und des großen Potenzials zur Vereinfachung von Geschäftsabläufen, Engineering und Anlagenbetrieb werden automatisch verarbeitbare Merkmalmodelle jedoch weit weniger häufig genutzt, als es sinnvoll und möglich wäre [31]. Insofern bedeutet die organisatorische und technische Unterstützung der genannten Standards längst noch nicht, dass damit alle elektronisch gespeicherten Merkmalinformationen zugreifbar und integriert sind. Eine breitere Unterstützung von standardisierten Formaten kann aber zukünftig erwartet werden, weshalb die aktuell aussichtsreichsten Kandidaten IEC 61360 und eCl@ass in den folgenden Abschnitten genauer betrachtet werden.

3.1.1. IEC 61360

Die Norm bzw. Normenreihe IEC 61360 trägt den deutschen Titel „Genormte Datenelementtypen mit Klassifikationsschema für elektrische Bauteile“. Im ersten Teil der Norm wird ein grundlegendes Metameta-Modell für merkmalbasierte Modellierung festgelegt, das dann im zweiten Teil auf das sogenannte EXPRESS-Datenmodell aus ISO 10303 [42] projiziert wird. Dadurch wird Kompatibilität mit der ISO 13584 [43] erreicht, die ebenfalls EXPRESS verwendet. Im vierten Teil der Norm werden schließlich typische Merkmale elektrischer Bauteile definiert, die auch online im „Common Data Dictionary“ eingesehen werden können [36]. An dieser Stelle ist aber hauptsächlich das im ersten Teil der Norm definierte Merkmalmodell von Interesse.

Datenelementtypen

Der „Datenelementtyp“ ist ein zentraler Begriff der IEC 61360. Er wird definiert als „Informationseinheit, deren Identifikation, Beschreibung und Wertdarstellung festgelegt sind“ [19]. Sinngemäß entsprechend der Verwendung des Begriffs handelt es sich bei einem Datenelementtyp um ein allgemeines Merkmal entsprechend der Terminologie aus

Abschnitt 2.1. Dem Datenelementtyp können zwar wie einem speziellen Merkmal Werte zugeordnet werden, wegen des Anwendungsbereichs der Norm für Katalogsysteme wird das aber als Aussage über alle Merkmalträger eines Typs verstanden.

Ein Datenelementtyp wird durch Attribute definiert, die sich auf „Identifikation, Beschreibung, Werte von Datenelementtypen und Beziehungen zwischen Datenelementtypen“ beziehen [19]. Die Norm listet insgesamt 25 solcher Attribute auf, die aber nicht alle obligatorisch sind. Einige davon haben einen direkten Bezug zu Elementen aus dem Merkmalmodell entsprechend Abschnitt 2.1, andere sind eher speziell für die IEC 61360. Hier sind natürlich die erstgenannten Attribute von Interesse.

Eine gewisse Menge von Attributen wird in der Norm als „identifizierende Attribute“ bezeichnet. Dies sind, vollständig aufgelistet, Kennung, Versionsnummer, Änderungsnummer, bevorzugter Name, Synonym, Kurzbezeichnung, bevorzugtes Formelzeichen und Synonym des Formelzeichens. Tatsächlich zur Identifikation eines Datenelementtyps notwendig ist aber nur die Kennung, die auch bei wesentlichen Änderungen wie Änderung der Definition neu vergeben wird. Bei Änderungen, die für die Verwendung des Datenelementtyps weniger schwerwiegend sind, wie Änderung des Formelzeichens, werden neue Versionsnummern oder Änderungsnummern vergeben. Insgesamt ist durch die Kombination aus Kennung, Versionsnummer und Änderungsnummer also eine eindeutige Identifikation eines bestimmten Datenelementtyps in einem bestimmten Zustand möglich. Die einzige Ausnahme bilden Kennungen, die mit dem Zeichen „X“ beginnen, weil diese nicht zentral vergeben werden und für lokal gebrauchte Datenelementtypen zur Verfügung stehen, z.B. für innerbetriebliche Zwecke.

Durch „semantische Attribute“ wird die Bedeutung eines Datenelementtyps erklärt. Das beinhaltet eine textuelle Definition, aber auch die Möglichkeit Formeln und Bilder einzuschließen.

„Wertattribute“ ordnen einem Datenelementtyp Werte zu. Der Begriff „Wert“ wird in der Norm nicht näher spezifiziert, entsprechend seiner Verwendung schließt er aber auch komplexe Werte ein, die sich aus mehreren Einzelwerten zusammensetzen (beispielsweise Vektoren). Weil Datenelementtypen allgemeinen Merkmalen entsprechen, kann durch Wertattribute nicht nur ein einzelner Wert, sondern eine Menge möglicher Werte hinterlegt werden. Mit dem Wert einher gehen Informationen zu dessen technischer und logischer Interpretation wie Datentyp, Codierung und Maßeinheit. Eine nennenswerte Variante des komplexen Wertes ist der „Niveauwert“, der für reelle und ganzzahlige Werte den Mindestwert, Nennwert, typischen Wert und Höchstwert in einem vierstelligen Vektor angibt. Mertens schreibt dazu [54]: „Hier versucht die Norm, in Ergänzung zur reinen Merkmalausprägung noch einen Teil der Semantik dieser Ausprägung explizit abzubilden.“ Entsprechend dem Modell aus Abschnitt 2.1 handelt es sich in dem Fall also um Ausprägungsaussagen, nicht nur um Werte. Ebenfalls erwähnenswert ist, dass entsprechend der Norm Wertattribute eine weitergehende Bedeutung für die Klassifikation des modellierten Objekts haben können (in dem Sinne, dass bestimmte Werte die Zugehörigkeit zu einer bestimmten Klasse bedingen), und dass es Abhängigkeiten zwischen Werten verschiedener Datenelementtypen geben kann.

Eine Zuordnung von Datenelementtypen zu Klassen von Datenelementtypen ist durch „relationale Attribute“ gegeben. Diese verweisen auf genau eine Klasse. Gegebenenfalls werden durch relationale Attribute auch wertmäßige Abhängigkeiten zu anderen Datenelementtypen ausgedrückt.

Klassen von Datenelementtypen

Die Spezifikation von Klassen von Datenelementtypen wird von der IEC 61360 deutlich weniger genau behandelt als die der Datenelementtypen selbst. Im Wesentlichen wird nur ausgesagt, dass durch die Bildung von Klassen die Handhabung größerer Mengen von Datenelementtypen vereinfacht werden soll. Dazu können baumförmige Hierarchien von Klassen gebildet werden, die sich auf quantitative und nicht-quantitative (z.B. Materialarten oder Bauformen) Datenelementtypen beziehen. Zu beiden Fällen gibt die Norm „Hauptklassen" vor, von denen sich weitere Klassen ableiten lassen. Wie die Spezifikation einer Klasse genau aussieht, wird aber nicht beschrieben.

Bauteilklassen

Wegen des Anwendungsbereichs der IEC 61360 wird dort die Spezifikation von Klassen elektrischer Bauteile festgelegt, prinzipiell können die Merkmalträger aber auch andere Objekte sein. Den Bauteilklassen werden Datenelementtypen zugeordnet. Zur Strukturierung wird eine baumförmige Hierarchie von Bauteilklassen verwendet, in der ein Vererbungsmechanismus für die Datenelementtypen von Klassen gilt, d.h. Datenelementtypen einer Klasse sind auch in hierarchisch untergeordneten Klassen enthalten. Die Kriterien zur Identifikation von Klassen werden nicht fest vorgegeben.

Bauteilklassen werden, wie Datenelementtypen, durch Attribute spezifiziert. In diesem Fall sind das identifizierende und semantische Attribute, die in Bedeutung und Verwendung den entsprechenden Attributen für Datenelementtypen sehr ähnlich sind. Bemerkenswerterweise werden die Datenelementtypen, die einer Bauteilklasse zugeordnet werden, nicht durch Attribute festgelegt. Überhaupt wird der Mechanismus dieser Zuordnung nicht explizit beschrieben. Die Darstellung in Diagrammen in Texten der Norm legt aber nahe, dass die Zuordnung eindeutig ist und dass anhand einer gegebenen Bauteilklasse die zugeordneten Datenelementtypen erkennbar sind.

Jede Bauteilklasse besitzt mindestens einen „klassifizierenden Datenelementtyp". Dieser Datenelementtyp unterscheidet die Bauteilklasse von ihrer hierarchisch übergeordneten Klasse, entweder durch eine Einschränkung der möglichen Werte, oder dadurch, dass der Datenelementtyp in der übergeordneten Klasse nicht vorkommt.

3.1.2. eCl@ss

eCl@ss ist ein Klassifikations- und Katalogsystem zur Unterstützung der technischen Materialwirtschaft in der chemischen Industrie, mit dessen Entwicklung 1997 begonnen wurde [27]. Das primäre Ziel von Klassifikations- und Katalogsystemen ist die Definition von Klassen von Merkmalträgern und von deren Merkmalen, so dass diese Merkmalträger (z.B. Geräte) in standardisiert aufgebauten Katalogen beschrieben werden können. Eine Festlegung der Art und Weise, wie die einzelnen Merkmalträger beschrieben werden, ist dagegen kein Primärziel, weshalb eCl@ss in dieser Hinsicht flexibel ist und beispielsweise auf die Strukturen der IEC 61360 abgebildet werden kann.

Hintergrund

Zunächst war eCl@ss ein Projekt von acht Chemieunternehmen, die Beschaffung, Lagerung, Reparatur, Dokumentation und Endverwertung von Geräten und Ersatzteilen unterstützen und vereinfachen wollten [27]. Im Jahre 2000 wurde dann der eCl@ss e.V. gegründet, der das Projekt weiter betrieb und ausbaute und nach eigenen Angaben im Jahr 2012 bereits ca. 120 Organisationen als Mitglieder zählte [26]. Durch die Kooperation mit anderen Katalogsystemen wurde das Anwendungsspektrum von eCl@ss auch auf Bereiche über die chemische Industrie hinaus erweitert. Aus Sicht der Prozessindustrie ist insbesondere die 2013 vorgenommene Eingliederung von Prolist bemerkenswert. Prolist bietet wie eCl@ss eine Klassifikation von Geräten, Listen ihrer Merkmale und dokumentierte Prozesse für deren Erstellung und Verwendung. Besondere Bedeutung hat Prolist, weil es durch die NAMUR-Empfehlung 100 [58] breiten Rückhalt der Anwender von Automatisierungstechnik in der Prozessindustrie hat.

Es werden regelmäßig neue Versionen von eCl@ss veröffentlicht; aktuell ist Version 9.0. Die Versionen unterscheiden sich einerseits in den enthaltenen Merkmalträgern, andererseits auch in dem verwendeten Metamodell in Bezug auf die Modellierung von Merkmalen (z.B. wird in Version 8.0 gegenüber 7.1 kein Formelzeichen mehr angegeben). Die Beschreibung bezieht sich hier auf Version 9.0.

Praxis

Die Geschäftsprozesse, in denen eCl@ss in der Prozessindustrie eingesetzt werden soll, werden in [31] beschrieben. Bild 3.1 gibt diese Prozesse schematisch wieder.

Der erste Schritt ist die Planung einer Produktionsanlage durch den Anlagenbetreiber oder einen beauftragten Dienstleister. Dieser spezifiziert Anforderungen an Geräte bzw. allgemein an das benötigte Material und schickt eine entsprechende Anfrage an Hersteller oder Lieferanten. Diese antworten darauf ggf. mit passenden Angeboten, so dass eine Bestellung und letztendlich auch Lieferung und Montage erfolgen können. Die Daten des gelieferten Materials werden in die Dokumentation der Anlage übernommen.

Später im Lebenszyklus der Anlage kann dann die Notwendigkeit entstehen, ein Gerät oder sonstiges Bauteil auszutauschen oder die Anlage zu erweitern. In diesem Fall wird seitens des Betreibers erneut das benötigte Material spezifiziert, wobei die vorhandene Dokumentation ein hilfreicher Ausgangspunkt ist. Zusätzlich zu den Anforderungen können so auch Informationen über das bisher eingesetzte Material an Hersteller oder Lieferant übermittelt werden. Dieser kann erneut ein Angebot formulieren und letztendlich, nach erfolgreicher Lieferung und Einbau, wird die bestehende Dokumentation aktualisiert.

Der geschilderte Ablauf kann auf unterschiedliche Art realisiert werden. Sogar ein manuelles Vorgehen ohne die Unterstützung durch elektronische Datenverarbeitung ist möglich. Typischerweise sind heute aber auf Seite des Anlagenbetreibers und auf Seite von Herstellern und Lieferanten entsprechende IT-Systeme vorhanden, so dass die Geschäftsprozesse innerhalb derselben Firma weitgehend automatisiert und effizient sind [31]. Der Datenaustausch zwischen den Firmen ist aber nur wenig automatisiert und von außen eingetroffene Daten müssen häufig manuell integriert werden. An dieser Stelle setzt eCl@ss an. Einerseits wird durch die standardisierte Klassifikation und die standardisierten Listen von Merkmalen eine Automatisierung dieser Prozesse logisch

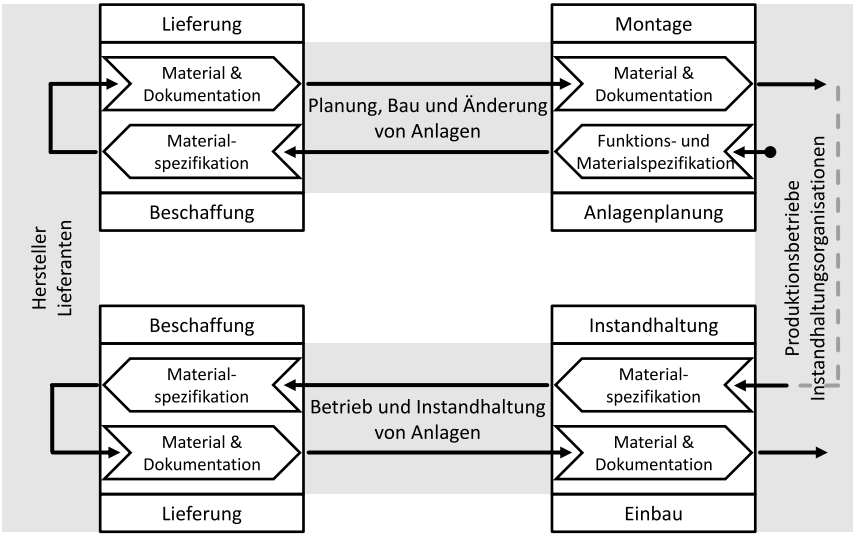

Abbildung 3.1.: Schemetische Darstellung der Geschäftsprozesse im Anwendungsbereich von eCl@ss nach [31].

möglich. Andererseits besteht durch die Abbildbarkeit auf Datenstrukturen (etwa der IEC 61360) die Möglichkeit der standardisierten technischen Repräsentation, z.B. durch XML-Dateien.

In Tabelle 3.1 wird gezeigt, welche Gewerke zu welchem Zeitpunkt eCl@ss einsetzen können. Diese Darstellung stellt die Anwendungsbereiche im Sinne der Zielsetzung von eCl@ss dar. Darüber hinaus ist aber auch eine Verwendung im Engineering der Anlage und in deren Betrieb möglich und erwünscht. In der NE 100 [58], deren Anwendungsziele kongruent mit eCl@ss sind, heißt es dazu: „Die NE 100 ermöglicht darüber hinaus, Gerätedaten als Merkmalleisten in Prozessleitsystemen oder Feldgeräten zu speichern." In [2] wird auch auf die gewinnbringende Verwendung von eCl@ass Daten bei Kombination mit Mitteln der Prozessbeschreibung wie dem Phasenmodell der Produktion [3] hingewiesen.

Struktur

Die Klassifikation in eCl@ss geschieht durch eine vierstufige Hierarchie. Jede Hierarchieebene hat eine eindeutige Kennnummer („kodierter Name"), die mit der Nummer der übergeordneten Ebene beginnt, so dass beispielsweise alle Einträge unterhalb des Sachgebiets „Maschine, Apparat" mit der Nummer 36 beginnen. Der Aufbau der Hierarchie wird in Tabelle 3.2 gezeigt. Die Bildung einer Klasse und deren Zuordnung zu einer übergeordneten Klasse der Hierarchie geschieht durch die Betrachtung technischer und kaufmännischer Aspekte [27].

Tabelle 3.1.: Anwendungsfälle von eCl@ss nach Lebenszyklusphasen einer Anlage und beteiligten Partnern nach [31].

	Beteiligte Partner				
Lebenszyklusphase	Planer	Betreiber/ Einkauf	Hersteller/ Vertrieb	Betreiber/ Instand-haltung	Hersteller/ After Sales Service
Planung: Anfrage und Angebot	•		•		
Planung: Bestellung	•	•	•		
Planung: Abschluss und Inbetriebnahme	•			•	
After Sales Service				•	•
Instandhaltung: Ersatzteilanfrage			•	•	
Instandhaltung: Ersatzteilbestellung		•	•	•	
Planung: Erweiterung	•			•	

Tabelle 3.2.: Die vier Klassifikationsebenen von eCl@ss mit je einem Beispiel.

Ebene	Name der Ebene	Bsp. kodierter Name	Bsp. bevorzugter Name
1	Sachgebiet	36-00-00-00	Maschine, Apparat
2	Hauptgruppe	36-41-00-00	Pumpe
3	Gruppe	36-41-01-00	Kreiselpumpe
4	Untergruppe	36-41-01-08	Kreiselpumpe mit Wellendichtung

Jede Untergruppe beinhaltet eine sogenannte „Merkmalleiste“, die praktisch eine Menge von allgemeinen Merkmalen ist. Dabei unterscheidet eCl@ss zwischen „Basismerkmalleisten“ und „Standardmerkmalleisten“. Die Basismerkmalleiste ist stets gleich und enthält daher nur allgemeine Merkmale, die immer anwendbar sind wie „Herstellername“ und „Artikelbezeichnung“. Sie wird dann verwendet, wenn entweder noch keine klassenspezifischen Merkmale festgelegt wurden oder wenn das nicht notwendig ist. Standardmerkmalleisten sind dagegen klassenspezifisch zusammengestellte allgemeine Merkmale.

Jedes Merkmal in einer Merkmalleiste wird durch einen eindeutigen Code identifiziert. Die Spezifikation von allgemeinen Merkmalen ist ebenfalls in eCl@ss enthalten und wird durch die Attribute „Abk.“, „Datenformat“, „Einheit“, „Definition“ und „Werte“ durchgeführt. Unter dem Attribut „Werte“ kann eine Liste von erlaubten Ausprägungen für ein Merkmal angegeben werden, wobei es hier auch vordefinierte und referenzierbare Ausprägungen gibt. Beispielsweise sind mögliche Bauformen von Kupplungen als Ausprägungen definiert. Als mögliche Datenformate ist die folgende Li-

ste vorgegeben: Boolean, String, String_Translatable, Integer_Count, Integer_Measure, Integer_Currency, Real_Count, Real_Measure, Real_Currency, Rational, Rational_Measure, Time, Timestamp, Date, Url.

3.2. Abfragesprachen

Für die Abfragen an Informationssysteme gibt es unterschiedliche Typen von Abfragesprachen. Diese richten sich nach der Struktur der für Abfragen verfügbaren Information, den Fähigkeiten des Informationssystems und der Anforderungen der Klienten. Im einfachsten Fall gibt es nur Operationen zum gezielten Auslesen von Information und ggf. auch zur Erstellung und Veränderung von Informationsobjekten. Im Bereich der Automatisierungstechnik gibt es einige Kommunikationsprotokolle entsprechend diesem Schema, beispielsweise OPC UA oder ACPLT/KS. Komplexe Informationssysteme wie Datenbanken unterstützen dagegen komplexere Abfragesprachen, z.B. auf Basis relationaler Algebra (siehe Abschnitt 2.2.2).

3.2.1. Abfragesprachen für relationale Datenbanken

Die theoretische Grundlage der Abfragesprachen für relationale Datenbanken ist die relationale Algebra. Eine frühe Umsetzung als praktisch anwendbare Abfragesprache war die Sprache SEQUEL [7], die später in SQL (oft interpretiert als Abkürzung für „Structured Query Language“) umbenannt wurde. SQL gilt heute als die Standardsprache für relationale Datenbanken [11] und ist normiert [45]. Unterschiedliche Implementierungen enthalten trotzdem oft zusätzliche Operationen, so dass auch von SQL-Dialekten gesprochen wird.

SQL implementiert viele der Operationen der relationalen Algebra direkt als einzelne Operation und ist relational vollständig. Die Sprache ist textbasiert, beispielsweise ist der Ausdruck „SELECT X FROM Y WHERE z“ ein häufiges Konstrukt, durch das aus der Relation Y die Attribute X derjenigen Tupel zurückgegeben werden, für die Bedingung z gilt. Es handelt sich also um die Verkettung einer Selektions- und Projektionsoperation. Komplexe Ausdrücke können durch Klammerung gebildet werden.

Wie in Abschnitt 2.2.2 diskutiert, deckt die relationale Algebra einige praktisch benötigte Anwendungsfälle nicht ab. SQL beinhaltet daher weitere Operationen, die diese Lücken größtenteils ausfüllen. Beispielsweise sind Aggregationsoperationen wie ORDER BY oder COUNT verfügbar.

3.2.2. Abfragesprachen für graphbasierte Datenbanken

Neben den vorherrschenden relationalen Datenbanken gibt es heute eine weitere verbreitete Art von Datenbanken, nämlich graphbasierte Datenbanken. In ihnen werden Daten nicht durch Tabellen, sondern durch ein Netz von Knoten und Kanten im Sinne eines gerichteten Graphs aus der Mathematik bzw. Informatik verwaltet. An die Knoten und Kanten können Attribute und Werte angehängt werden; außerdem sind die Kanten typisiert. Üblicherweise repräsentieren die Knoten modellierte Objekte (entsprechend wie Tupel im relationalen Modell) und die Kanten Beziehungen zwischen den Objekten. Anders als bei relationalen Datenbanken hat sich kein Standard für Abfragesprachen

herausgebildet. Die vorhandenen Sprachen haben lediglich die Graphentheorie als gemeinsame Grundlage.

Abhängig von der Struktur der Daten ist die Geschwindigkeit in der Verarbeitung von Abfragen teilweise deutlich größer als bei relationalen Datenbanken [76]. Diese Beobachtung trifft insbesondere dann zu, wenn die interne Verknüpfung von Daten durch Kanten des Graphen genutzt werden kann. Andererseits arbeiten relationale Datenbanken auf großen, homogen strukturierten Datensätzen prinzipbedingt effizienter.

3.2.3. Domänenspezifische Abfragesprachen

SQL sowie die unterschiedlichen Sprachen für graphbasierte Datenbanken sind als allgemeine Abfragesprache konzipiert. Es gibt daher keine Annahmen über die Semantik der abgefragten Daten. Die Konstruktion von speziellen Abfragesprachen mit gewissen Annahmen über die Semantik ist in der Literatur und Praxis eine Ausnahme, einige Beispiele sind jedoch vorhanden. Üblicherweise wird dabei die relationale Algebra als Grundlage verwendet, um spezielle Operatoren mit spezieller Semantik zu definieren. Beispielsweise existiert eine genormte Erweiterung von SQL, die den Umgang mit speziellen Daten wie Bildern oder Text vereinfachen soll [44]. Eine formale Abfragesprache für Dokumentinhalte wird von Mhlanga et al. vorgeschlagen [56]. Sparr präsentiert eine Abfragesprache für relationale Datenbanken mit Erweiterungen speziell für den Anwendungsbereich CAD [68]. Allgemein gibt es im Anwendungsbereich der Automatisierung aber eine viel stärkere Tendenz hin zu standardisierten Dateiformaten für den Informationsaustausch, während spezielle Abfragesprachen kaum diskutiert werden. Vor diesem Hintergrund ist es sehr interessant zu sehen, dass Barth und Fay die Abfragesprache LINQ, eine SQL-ähnliche Abfragesprache innerhalb der Programmiersprache C# , zur Abfrage von Information aus Engineering-Dateien vorschlagen [5]. Die Autoren stellen zwar keine spezielle Abfragesprache für Engineering-Daten vor, betonen aber die möglichen Vereinfachungen und Geschwindigkeitszuwächse bei Verwendung gezielter Abfragen gegenüber der üblichen Verarbeitung gesamter Dateien.

3.3. Software-Systeme im Umfeld der industriellen Produktion

In der industriellen Produktion werden zahlreiche Software-Systeme verwendet, die Informationen über technische Merkmale nutzen und verwalten. Dieser Abschnitt gibt einen Überblick über die wichtigsten dieser Systeme. Ziel ist es, solche Systeme zu identifizieren, in denen Information zu Merkmalen vorliegt, die für ein Automatisierungssystem operativ nutzbar ist. Die technische Realisierung dieser Systeme ist dabei nachrangig.

Generell gibt es im Umfeld der industriellen Produktion keine einheitliche Standardsoftware. Stattdessen sind der Umfang und die Art der einzelnen Software von Betrieb zu Betrieb unterschiedlich. Selbst wenn das gleiche Software-System in der gleichen Version in zwei Betrieben verwendet wird, dann ist die individuelle Konfiguration, beispielsweise durch die Installation von zusätzlichen Erweiterungen, praktisch immer verschieden. Insofern wäre es unsinning, an dieser Stelle Aussagen über konkrete Software-

Systeme zu treffen.

Eine weit verbreitete Sicht auf die produktionsrelevanten Software- und Automatisierungssysteme ist die Einteilung in Ebenen, die jeweils einem Aufgabenbereich entsprechen („Automatisierungspyramide“, siehe Abbildung 5.6, Seite 78). Dieses Modell umfasst zumindest die Prozessleitebene (unten), die Produktionsleitebene und die Unternehmensleitebene (oben) [70]. Zwischen Prozessleitebene und Produktionsleitebene wird z.T. auch eine „Betriebsleitebene“ genannte Zwischenebene aufgezählt und je nach Autor und Zielsetzung kann die Ebene der Feldgeräte und auch der Prozess selbst als Ebene enthalten sein.

Die Beziehung zwischen den Ebenen der Automatisierungspyramide sah ursprünglich so aus, dass von oben nach unten Vorgaben für die Produktion gemacht werden und von unten nach oben Daten aus der Produktion weitergereicht und verdichtet werden [70]. Durch die Möglichkeiten netzwerkbasierter Kommunikation, in der nicht nur fest konfigurierte Punkt-zu-Punkt Kommunikation möglich ist, und durch darauf aufbauende Diensttechnologien, zeichnet sich aber schon seit einigen Jahren eine Flexibilisierung der Informationsflüsse ab (siehe z.B. [51], [65]). Daher behält die Zuteilung eines Software-Systems zu einer der Ebenen anhand seiner Aufgabe zwar ihre Gültigkeit, die Zuteilung begrenzt aber nicht die möglichen Informationsflüsse. Im Kontext dieser Arbeit bedeutet das, dass für die Relevanz eines Software-Systems nicht die hierarchische Einordnung, sondern dessen Aufgabe und damit die genutzte Information entscheidend sind.

Die Aufgaben der Software-Systeme setzen sich aus einigen Kernaufgaben zusammen, die in der industriellen Produktion grundsätzlich anfallen. In der ANSI-Norm ISA-95 bzw. deren Pendant IEC 62264 [39] mit dem Titel „Integration von Unternehmensführungs- und Leitsystemen“ wird ein wesentlicher Teil dieser Aufgaben ausführlich aufgelistet und erläutert. Eine Auswahl daraus ist (im Wortlaut der Norm [39]):

- Unternehmensleitebene
  - „Erfassung und Pflege des Rohstoff- und Ersatzteilverbrauchs sowie des verfügbaren Bestands und Bereitstellung von Daten für den Einkauf von Rohstoffen und Ersatzteilen;“
  - „Erfassung und Pflege aller Waren in Bestandsdateien für Prozess und Produktion;“
  - „Erfassung und Pflege von Aufzeichnungen über die Nutzung der Maschinen und Ausrüstungen und der Laufzeithistorie, die für die vorbeugende und vorausschauende Instandhaltungsplanung erforderlich sind;“
  - „Planung einer optimalen vorbeugenden Instandhaltung und Erneuerung der Ausrüstung in Verbindung mit dem grundlegenden Produktionsplan für die Anlage;“
  - „Bestimmung der optimalen Bestände von Rohstoffen, Energien, Ersatzteilen und Waren im Prozess an jedem Lagerort [...];“
- Produktionsleitebene

- „Erfassung und Pflege von Daten über Produktion, Bestand, Personal, Rohstoffe, Produktqualität, Ersatzteile und Energieverbrauch im Betriebskomplex;“
- „Datenerfassung und Offline-Datenanalyse gemäß Vorgabe durch Engineering-Funktionen; dies kann eine statistische Qualitätsanalyse und zugehörige Steuerungsfunktionen einschließen;“
- „Verwaltung der Instandhaltung der Produktionsausrüstung;“
- „Verwaltung der Labor- und Qualitätsprüfungen an Materialien;“
- „Verwaltung von Transport und Lagerung der Materialien; [...]“

Neben diesen Aufgaben werden zahlreiche weitere genannt, die aber keine Information enthalten, die für ein Automatisierungssystem in der Produktion sinnvoll nutzbar wäre (beispielsweise Personalplanung).

Kommerzielle Software-Systeme zur Bewältigung der genannten Aufgaben sind verfügbar, allerdings besteht nicht immer eine eins-zu-eins Beziehung zwischen Aufgabe und Software-System. Oft ist es möglich, mit einem System mehrere Aufgaben abzudecken. Außerdem ist das Ziel der ISA-95 lediglich die Aufgaben, Vorgänge und Begriffe der Software-Systeme festzulegen. Die Implementierung der technischen Schnittstellen und der Datenrepräsentation wird nicht spezifiziert und ist daher aus Sicht der Norm zunächst proprietär. Dadurch klärt die Norm zwar, welche Information in einem Software-System vorliegt, aber nicht, wie darauf zugegriffen werden kann. Für die Vereinfachung und Vereinheitlichung der Integration von ISA-95-konformer Software gibt es daher separate Ansätze. Die Organisation MESA (Manufacturing Enterprise Solutions Association) hat mit der „Business To Manufacturing Markup Language (B2MML)“ ein XML-Datenformat veröffentlicht, das den Austausch von Information entsprechend den ISA-95-Informationsmodellen ermöglicht [53]. Weiterhin existiert eine Spezifikation der OPC Foundation für die Abbildung der ISA-95-Modelle im Kommunikationsstandard OPC UA [15, 59]. Beide Ansätze zeigen, dass die Modelle der ISA-95 große Bedeutung haben und dass zumindest seitens der Softwareanwender starkes Interesse an einer Vereinfachung der Datenanbindung vorliegt. Letztendlich liegt es aber bei den Softwareherstellern, diese Standards zu unterstützen. Aktuell gibt es daher in der Praxis keine einheitlichen Datenschnittstellen der Software-Systeme.

Die folgenden Abschnitte geben einen Überblick über die wichtigsten und am weitesten verbreiteten Arten von Software-Systemen im Umfeld der industriellen Produktion.

3.3.1. Manufacturing Execution Systeme

Manufacturing Execution Systeme (MES) umfassen oft mehrere der genannten Aufgaben in der Produktions- und Betriebsleitebene. Im Arbeitsblatt 94 der NAMUR [57] wird dazu festgestellt, dass „MES“ oft auch als Bezeichnung der Betriebsleitebene verwendet wird, obwohl MES-Softwarepakete nicht die gesamte Funktionalität der Betriebsleitebene abdecken. Daher „ist der Term MES mit Vorsicht zu benutzen bzw. in seiner jeweils gedachten Funktionalität zu spezifizieren.“ [57] Auch die VDI-Richtlinie 5600 zum Thema MES [74] enthält keine abgeschlossene Beschreibung der Aufgaben, es werden aber in Übereinstimmung mit dem Arbeitsblatt der NAMUR die folgenden Aufgaben genannt:

- Produktionsfeinplanung
- Materialflusssteuerung
- Qualitätsmanagement
- Bestandsführung
- Produktionsdokumentation

Durch die Erfüllung dieser Aufgaben sind in MES viele und nützliche Informationen zu Merkmalen vorhanden. Beispiele sind: Belegungszustände von Produktionsressourcen, Merkmale von Produktionsaufträgen, Materialbestände und -lagerorte, statistische Werte zur Produktqualität und zur Produktionsmenge. Diese Information kann in der operativen Automatisierungstechnik genutzt werden, um die Produktqualität zu verbessern, Produktionskosten zu senken und Produktionsmengen zu erhöhen.

3.3.2. Datenarchive

Die Archivierung von Daten aus der Produktion kann auch als Teilfunktion eines MES vorliegen; praktisch kommt hierfür aber oft ein eigenständiges System zum Einsatz. Hauptziele dieser Systeme sind das sichere und langfristige Speichern von Umfangreichen Daten, die zumeist direkt, d.h. mittels Sensoren, in der Produktion erhoben werden. Dies kann der Erfüllung rechtlicher Rahmenbedingungen dienen, z.B. in der Pharmazie- und Lebensmittelproduktion, ist aber auch eine Grundlage zur allgemeinen Qualitätsüberwachung und Fehlerdiagnose. Bedeutung erhalten Datenarchive auch durch die Möglichkeit der automatisierten Analyse der Daten, die entweder unter Einbeziehung von Wissen über den Prozess und die Produktionsanlage geschehen kann, oder ohne dieses Wissen mit Verfahren des „Data Mining“ [66]. Nützliche Merkmalinformation in Datenarchiven betrifft insbesondere historische Messwerte und Messwerte, aggregierte Informationen und Information die von Systemen stammt, zu denen es keine direkte Kommunikationsverbindung gibt.

3.3.3. Rezeptverwaltung

Die Aufgabe und Funktion von Systemen zur Rezeptverwaltung wird in der Normenreihe DIN EN 61512 „Chargenorientierte Fahrweise“ [14], der deutschen Version der ANSI ISA 88, festgelegt. Entsprechend handelt es sich dabei um ein Software-System, das in der Prozessindustrie bei chargenweiser Produktion Anwendung findet.

In einem System zur Rezeptverwaltung werden Produktionsrezepte auf vier Abstraktionsebenen verwaltet. In der Norm umfasst die Rezeptverwaltung das grundsätzlichen Verfahrensrezept, das unabhängig von der Anlagenausrüstung ist, das Werksrezept, das Grundrezept und schließlich das Steuerrezept, das spezifisch für eine bestimmte Anlage und eine bestimmte Charge aus dem Grundrezept erstellt wird. Neben der eigentlichen Prozedur werden in den Rezepten Stoff- und Produktionsparameter festgelegt. Sie umfassen die notwendigen Grundstoffe des Prozesses (Prozesseinsatz) in Menge und Art, die durch den Prozess erzeugten Stoffe und Energien (Prozessausstoß) und Prozessparameter wie Sollwerte und Vergleichswerte zu messbaren Größen des Prozesses. Außerdem können in Rezepten Anforderungen an die Produktionseinrichtung wie zulässige

Werkstoffe hinterlegt werden. Somit sind neben der Prozedur auch operativ relevante Merkmale in Systemen zur Rezeptverwaltung vorhanden.

3.3.4. Labor-Informations- und Managementsysteme

Labor-Informations- und Management-Systeme, abgekürzt LIMS, haben den Haupteinsatzzweck, in Laboren ermittelte Daten zu speichern und zu analysieren. Im Kontext der industriellen Produktion betreffen diese Daten meist die Ausgangstoffe, Zwischenprodukte oder Endprodukte des Produktionsprozesses. Daher sind in einem LIMS Informationen zu Merkmalen vorhanden, die nicht durch Messung im Prozess ermittelt werden können, die den Prozess aber direkt betreffen.

3.3.5. Condition Monitoring

Unter „Condition Monitoring“ wird hier die „Zustandsüberwachung und -diagnostik von Maschinen auf der Basis von Parametern wie Schwingungen, Temperatur, Durchflussraten, Verunreinigung, Leistung und Drehzahlen, die typischerweise in Zusammenhang mit Funktion, Zustand und Qualitätskriterien stehen“ [20], verstanden. Entsprechende Software-Systeme beziehen die notwendigen Daten einerseits aus statischen Maschinendaten und andererseits aus Messungen des Produktionsprozesses oder der Maschinen. Ziel ist die frühzeitige Erkennung von Fehlern und Defekten sowie die Unterstützung von Wartungsaufgaben. Condition Monitoring-Systeme verwalten daher Informationen zu Merkmalen, die die Anlagenausrüstung und nicht den Produktionsprozess betreffen.

3.4. SPS-Programmierung

In der verfahrenstechnischen Industrie bilden Prozessleitsysteme den Kern der heute üblichen Automatisierungslösungen und innerhalb dieser werden meist speicherprogrammierbare Steuerungen (SPS) für die operative Ausführung der Mess- Steuer- und Regelaufgaben verwendet [71]. Die genaue Abgrenzung von SPS und Prozessleitsystem ist aber schwierig: Durch die Integration der SPS in das Prozessleitsystem verwischt die Unterscheidung zwischen dem Leitsystem und der SPS zunehmend [71]; außerdem werden SPS auf sehr unterschiedlichen Hardwaresystemen implementiert. An dieser Stelle wird daher nicht genauer auf die begriffliche Abgrenzung von SPS oder deren technische Realisierung eingegangen. Wichtiger ist hier, wie eine SPS programmiert wird und wie die Programme ausgeführt werden, d.h. die funktionale Sicht auf Engineering und Laufzeitverhalten. SPS-Hersteller wenden dazu in praktisch allen Fällen die Normenreihe IEC 61131 (bzw. DIN EN 61131 [18]) an. Aus sprachlichen Gründen werden hier die Begriffe der DIN EN 61131 verwendet.

3.4.1. Aufbau und Funktionsweise einer SPS

Gemäß IEC 61131 kann eine SPS in die in Bild 3.2 gezeigten funktionalen Einheiten unterteilt werden. Neben der funktionalen Unterteilung der Einheiten werden auch die internen und externen Schnittstellen gezeigt, über die Information (bzw. Energie im Falle

der Stromversorgung) übertragen wird. Von Interesse sind hier insbesondere die Signalverarbeitungsfunktionen. Diese werden im dritten Teil der Norm [21] genauer erläutert. Die einzelnen Signalverarbeitungsfunktionen werden als „Ressourcen“ bezeichnet. Zu ihnen gehören ggf. Schnittstellen zum Bediener, Sensoren und Aktoren. Die Funktionalität einer Ressource ist in einem Programm oder mehreren Programmen implementiert. Die Ausführung dieser Programme geschieht durch „Tasks“, die ebenfalls Teil der Ressource sind. Die Daten, mit denen ein Programm arbeitet, werden in Variablen innerhalb der Programme abgelegt. Auf diese Variablen können andere Programme nicht zugreifen. Darüber hinaus gibt es einen globalen Speicherbereich, in dem Variablen verwaltet werden, die von unterschiedlichen Ressourcen und Programmen aus erreichbar sein sollen.

Tasks besitzen je eine Liste von Programmen, die nacheinander bei Eintreten eines definierten Ereignisses ausgeführt werden. Dieses Ereignis ist häufig periodisch und durch einen internen Zeitgeber ausgelöst, es kann aber auch spontan eintreten wie z.B. die Änderung eines Sensorwerts. In welcher Reihenfolge die Programme von einer Task aufgerufen werden, wird durch die Norm nicht definiert. Entsprechend wird dies in der Praxis zwischen unterschiedlichen SPSen nicht einheitlich gehandhabt und kann zu Laufzeitunterschieden für dieselbe Ressource auf unterschiedlichen Systemen führen. Üblicherweise kann die Ausführungsreihenfolge aber vom Programmierer einer SPS angepasst werden.

Auf Ebene der Ressource kann die Ausführung gestartet und gestoppt werden. Nach dem Start werden die Variablen innerhalb einer Ressource initialisiert und alle enthaltenen Tasks gestartet. Beim Stoppen werden entsprechend die Tasks angehalten.

3.4.2. Programmiersprachen

Programme werden typischerweise in einer der fünf Programmiersprachen der IEC bzw. DIN EN 61131-3 geschrieben. Diese sind (deutsch/englisch): Anweisungsliste/Instruction List (AWL/IL), Strukturierter Text/Structured Text (ST/ST), Kontaktplan/Ladder Diagram (KOP/LD), Funktionsbausteinsprache/Function Block Diagram (FBS/FBD) und Ablaufsprache/Sequential Function Charts (AS/SFC). Neben der Norm selbst existiert der technische Bericht IEC TR 61131-8, „Guidelines for the application and implementation of programming languages“ [40], in dem weitere Details zur Implementierung der Sprachen erläutert werden.

Neben den Sprachen der IEC 61131 gibt es oft die Möglichkeit, Programme zu importieren, die in einer anderen Sprache wie C oder C++ geschrieben wurden. Dies ist aber eher als Speziallösung für besondere Anwendungsfälle zu sehen und entspricht nicht dem normalen Vorgehen für die Programmierung einer SPS.

AWL und ST sind textuelle Programmiersprachen, in denen Befehle zeilenweise geschrieben und ausgeführt werden. Die Sprachen AS, KOP und FBS sind graphische Programmiersprachen und unterscheiden sich daher schon durch ihre Repräsentation deutlich von AWL und ST (wobei es für AS auch die Möglichkeit der textuellen Programmierung gibt). Die folgenden Abschnitte geben einen kurzen Überblick über die Programmiersprachen. Anschließend wird auf die Sprache FBS im Detail eingegangen.

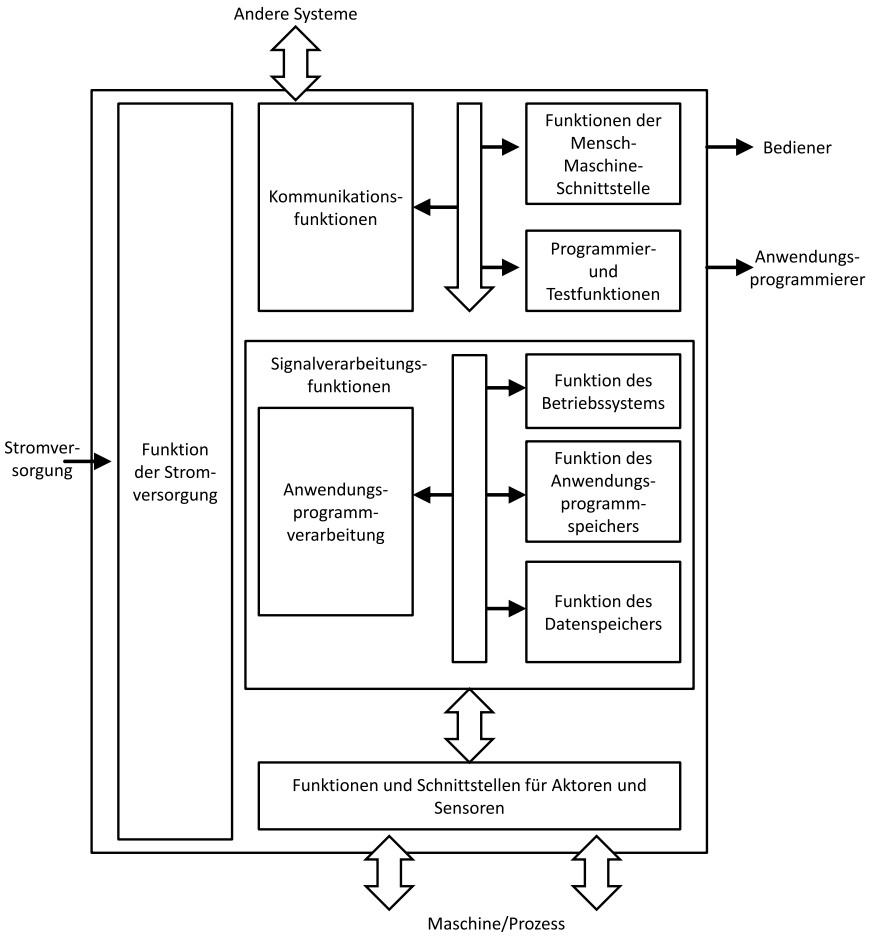

Abbildung 3.2.: Funktionale Grundstruktur einer SPS nach DIN EN 61131 [18].

Anweisungsliste und Strukturierter Text

AWL ist syntaktisch einfach aufgebaut: Eine Zeile beginnt mit einer Sprungmarke, dahinter stehen ein Operator bzw. eine Funktion, gefolgt von den Operanden. Die Operatoren und Funktionen sind verhältnismäßig maschinennah und einfach, beispielsweise sind LD (lade Bit), AND (logisches $\wedge$) und JMPCN (bedingter Sprung) typische Befehle. AWL wird häufig als gemeinsame Zwischensprache zwischen den anderen Programmiersprachen verwendet [47]. ST bietet vergleichsweise komplexere Sprachelemente wie FOR- und WHILE-Schleifen und ist daher mit höheren Programmiersprachen wie Pascal vergleichbar. Hauptvorteil von ST gegenüber AWL ist daher die bessere Kompaktheit und Einfachheit der Programme, während die Programme andererseits weniger effizient sein können und der Programmierer weniger Details unter eigener Kontrolle hat [47].

Ablaufsprache

AS ist eine Sprache zur Programmierung von Abläufen durch diskrete Zustände. Beispielsweise könnte die Steuerung für ein Gerät, das die möglichen Zustände „ein“ und „aus“ besitzt, grundsätzlich durch diese beiden Zustände – in der Terminologie von AS „Schritte“ genannt – programmiert werden. Bei der Ausführung des Programms werden diese Schritte nacheinander aktiviert. Zwischen den Schritten gibt es Transitionen, die durch Bedingungen überwachen, wann ein Schritt beendet wird und dessen Nachfolger aktiviert wird. Diese Bedingungen werden durch Boolesche Algebra formuliert und nehmen auf Variablen der SPS Bezug. Innerhalb der Schritte können Aktionen ausgeführt werden, die ihrerseits selbst IEC 61131-3 -Programme sind, und es können Variablen beschrieben werden. Für die Ausführung ist es notwendig, dass genau ein Schritt als Anfangsschritt markiert ist und dass es entweder mindestens einen Endschritt gibt, oder dass die Transitionen eine Rückwärtsschleife bilden. In diesem Fall hat der Ablauf kein spezifiziertes Ende.

Kontaktplan und Funktionsbausteinsprache

KOP und FBS haben ihren Ursprung in der Nachbildung von elektrischen Schaltungen und erlauben daher die Programmierung durch Erstellung eines Netzwerks von Elementen (KOP) bzw. Funktionsbausteinen (FBS). Jedes Element und jeder Funktionsbaustein besitzt (Signal-) Eingänge und Ausgänge, deren funktionale Beziehung das Element bzw. der Baustein herstellt. Im Fall von KOP entsprechen die Signale meist Booleschen Werten, während bei FBS alle Arten primitiver Datentypen verwendet werden können. FBS hat sich daher als universell einsetzbare Sprache etabliert [47].

Ausgänge werden paarweise durch Signallinien mit Eingängen verknüpft, so dass ein Netzwerk von Elementen bzw. Funktionsbausteinen entsteht. Der Wert eines Signals wird dann entlang der Signallinie von einem Ausgang zum Eingang „transportiert“. Die Verknüpfung von Ein- und Ausgängen mit Variablen der SPS ist ebenfalls möglich.

In Bild 3.3 wird ein einfaches Beispiel für ein FBS-Programm gezeigt. Es berechnet $Z = 3 \cdot X \cdot Y + 1$, wobei X, Y und Z für das Programm lesbare bzw. schreibbare Variablen sind. Zu jedem Baustein wird ein lokal eindeutiger Bezeichner (z.B. „add1“) und dessen Typ („ADD“) angegeben. Die Eingänge eines Bausteins befinden sich entsprechend der Norm auf der linken Seite und die Ausgänge auf der rechten und sie besitzen jeweils

einen eindeutigen Bezeichner innerhalb des Bausteins. Eine exakte Vorgabe über das Aussehen der Bausteine oder des Diagramms wird in der IEC 61131 aber nicht gemacht. Einige wichtige Bausteintypen werden in der VDI/VDE-Richtlinie 3696 [73] definiert, indem dort Typ, Eingänge, Ausgänge und Funktionalität definiert sind. Die Bausteine in Bild 3.3 sind dieser Richtlinie entnommen.

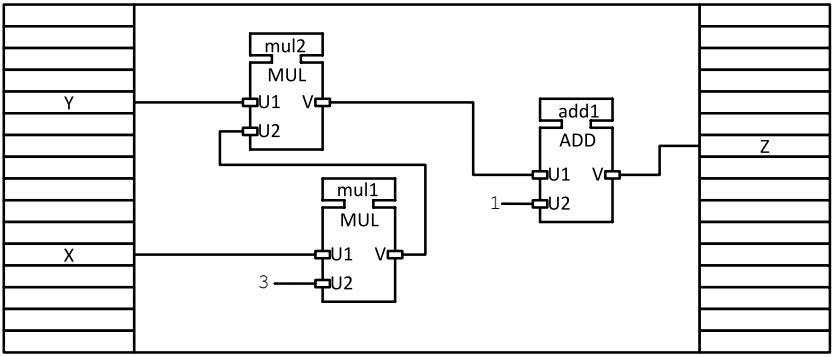

Abbildung 3.3.: Beispiel für ein Funktionsbausteinnetzwerk, das $Z = 3 \cdot X \cdot Y + 1$ berechnet.

Eigenschaften der IEC 61131-3 Programmiersprachen

Weil innerhalb einer SPS auch zeitgleich unterschiedliche Sprachen verwendet werden können, ist eine Kombination möglich und oft auch praktisch sinnvoll. Einfache Funktionen lassen sich mit AWL und KOP realisieren, komplexere Funktionen mit ST und FBS. Für übergeordnete Abläufe, die diese Funktionen nutzen, bietet sich AS an.

Für die Auswahl einer Sprache zur Lösung eines (Teil-)Problems sind Größe und Art des Problems einerseits und Eigenschaften der Programmiersprache andererseits entscheidend. Neben dem offensichtlichen Unterschied der graphischen oder textuellen Repräsentation eines Programms gibt es auch Unterschiede im zugrunde liegenden Programmierparadigma. Hier ist insbesondere die Unterscheidung zwischen imperativen und deklarativen Sprachen sinnvoll.

Bei imperativen Sprachen werden dem Rechner nacheinander Befehle gegeben, die in dieser Reihenfolge ausgeführt werden. Praktisch können die Befehle vom Compiler oder Prozessor anders sortiert werden, das geschieht aber immer ohne Beeinträchtigung der Semantik. Imperative Sprachen entsprechen daher einer prozeduralen Denkweise, in der der zeitliche Ablauf der Ausführung von Bedeutung ist. Beispielsweise kann

$$X' = 3 \cdot X; Y' = X' \cdot Y;$$

zu einem anderen Ergebnis führen als

$$Y' = X' \cdot Y; X' = 3 \cdot X;$$

weil einmal auf den „neuen“ Wert $(3 \cdot X)$ von X‘ zugegriffen wird und einmal auf den „alten“, initialen Wert. AWL, ST und AS sind imperative Sprachen. Sie eignen sich dann, wenn eine Aufgabe in zeitdiskreten Schritten bzw. durch Definition von diskreten Zuständen gelöst wird.

Im Vergleich dazu werden bei einer deklarativen Programmiersprache die genannten Befehle als zeitlich unabhängige Deklaration der Abhängigkeiten zwischen den Variablen interpretiert. Entsprechend wären die oben genannten Befehlsfolgen äquivalent, weil sie sich in beiden Fällen zu

$$Y' = 3 \cdot X \cdot Y$$

vereinfachen lassen. Deklarative Programmiersprachen sind daher dann besonders geeignet, wenn zeitinvariante Beziehungen zwischen Variablen bestehen, beispielsweise für Regler oder Simulationen. Ein Programm kann dann als Modellierung dieser Beziehungen betrachtet werden. Deklarative Sprachen sind KOP und FBS.

Praktisch werden auch Programme in deklarativen Programmiersprachen von einem Prozessor ausgeführt, der in zeitdiskreten Schritten arbeitet. Sofern diese Zeitschritte klein genug sind, kann ein Programmierer aber von einem „quasi-kontinuierlichen“ Ablauf ausgehen. Üblicherweise lässt sich die Größe dieser Zeitschritte auch vom Anwender festlegen. Bei FBS besteht zusätzlich auch die Möglichkeit, Funktionsbausteine gezielt einzeln aufzurufen. Beispielsweise kann das durch einen Funktionsaufruf in einem ST-Programm geschehen. In dem Fall wird die Berechnungsfunktion des Bausteins einmalig aufgerufen und dessen Ausgänge werden aktualisiert. FBS kann daher auch für zeitdiskrete Probleme verwendet werden, was je nach Anwendungsgebiet (z.B. in der diskreten Fertigung) mehr oder weniger üblich ist.

Tabelle 3.3 gibt einen Überblick über die Eigenschaften der Programmiersprachen der IEC 61131-3.

Tabelle 3.3.: Programmiersprachen der IEC 61131-3 und deren Eigenschaften.

	Repräsentation		Paradigma		Anwendung	
Sprache	graphisch	textuell	imperativ	deklarativ	kontinuierlich	zeitdiskret
AWL		•	•			•
ST		•	•			•
AS	•		•			•
KOP	•			•	•	
FBS	•			•	•	(•)

Spezifikation von Funktionsbausteintypen

Gemäß der IEC 61131-3 werden Funktionsbausteintypen durch bestimmte Schlüsselworte der Sprache ST deklariert. Diese Typen werden dann in Funktionsbausteinnetzen ggf. mehrfach als Funktionsbausteine instanziiert. Jeder einzelne Funktionsbaustein besitzt einen eigenen eindeutigen Namen und arbeitet mit einem eigenen Satz Ein- und Ausgabevariablen. Der ausgeführte Algorithmus ist aber für alle Bausteine eines Typs identisch. Der ST-Code in Codelisting 3.1 deklariert beispielhaft den Funktionsbausteintyp ADD.

```
FUNCTION_BLOCK ADD
VAR_INPUT
        (* Eingaenge des Bausteins *)
        U1 : REAL := 0.0;
        U2 : REAL := 0.0;
END_VAR
VAR_OUTPUT
        (* Ausgaenge des Bausteins *)
        V  : REAL := 0.0;
END_VAR
VAR_IN_OUT
        (* Als Eingang und Ausgang verwendbar *)
END_VAR
VAR
        (* Interne Variablen *)
END_VAR

(* Algorithmus *)
V := U1 + U2;

END_FUNCTION_BLOCK
```

Listing 3.1: ST-Quellcode den Funktionsbausteintyps ADD.

Ausführungssemantik von FBS

Prinzipiell ist FBS eine deklarative Programmiersprache und in der Theorie ist die Semantik eines Funktionsbausteinnetzwerks durch eine eindeutige Funktion definiert. Diese Funktion kann durch Verknüpfung der Funktionsbausteine entsprechend der Signallinien gebildet werden. Im Beispiel aus Bild 3.3 ist diese Funktion

$$Z = f(X,Y) = ADD(MUL(Y,MUL(X,3)),1).$$

Praktisch werden Programme einer SPS aber von einem Mikroprozessor ausgeführt, der in diskreten Zeitschritten arbeitet und dem imperativen Paradigma folgt. Daher gibt es praktisch natürlich eine Ausführungsreihenfolge für das Netzwerk und diese Reihenfolge hat Auswirkungen auf die Semantik.

Die Ausführungssemantik von FBS wird in der IEC 61131-3 nicht im Detail festgelegt, sondern es werden nur einige grundsätzliche Regeln definiert. Dazu gehört beispielsweise, dass ein Funktionsbaustein erst ausgewertet werden darf, wenn die Zustände aller Eingänge feststehen. Entsprechend darf die Auswertung des Funktionsbaustein(netzwerks) erst abgeschlossen werden, wenn die Zustände aller Ausgänge ermittelt wurden. Die Definition der Ausführungsreihenfolge muss durch den Programmierer geschehen, indem die in Abschnitt 3.4.1 beschriebenen Tasks festgelegt werden. Dazu werden die Frequenz der Ausführung und die Priorität einer Task angegeben; anschließend können Funktionsbausteine den Tasks zugewiesen werden. So kann die Ausführungsreihenfolge für Bausteine in unterschiedlichen Tasks festgelegt werden. Die Defi-

nition der Reihenfolge innerhalb einer Task soll entsprechend der Norm durch „herstellerspezifische Mittel“ geschehen. Praktisch werden dazu meist Tasklisten verwendet.

Eine alternative Methode der Ausführungskontrolle ist die ereignisgesteuerte Ausführung, die in der Normenreihe IEC 61499 bzw. DIN EN 61499 [22] festgelegt wird. Darin erzeugt die Ausführung eines Bausteins ein Ereignis, durch das die Ausführung anderer Funktionsbausteine ausgelöst werden kann. Dadurch ist es möglich, Bausteine „nur bei Bedarf“ auszuführen. Durch die zyklische Generierung von Ereignissen kann das zyklische Ausführungsmodell der IEC 61131-3 nachgebildet werden. Letztlich muss aber auch hier der Benutzer die Reihenfolge indirekt durch die Festlegung der Ereignisse definieren.

3.4.3. Kommunikation

Eine SPS besitzt mehrere Datenschnittstellen: Zum gesteuerten Prozess, zum Bediener, für die Programmierung und zu anderen Systemen (siehe Bild 3.2). Im Kontext dieser Arbeit ist die Schnittstelle zu anderen Systemen interessant, weil hier ein generischer, d.h. nicht zweckmäßig vorbestimmter Datenaustausch des SPS-Programms stattfindet. Die Möglichkeiten der Kommunikation von System zu System hängen in erster Linie vom Hersteller einer SPS ab. Es gibt aber auch standardisierte bzw. genormte und damit offen dokumentierte Kommunikationsschnittstellen und -protokolle.

OPC UA

OPC Unified Architecture (OPC UA) ist ein Kommunikationsstandard für die industrielle Anwendung. Er wurde vom Konsortium OPC Foundation erarbeitet und von der IEC als Norm veröffentlicht [15]. Die darin beschriebene Technologie besteht aus einem Client-Server-Modell, in dem der Klient Dienste des Servers aufrufen, Daten abfragen und manipulieren kann. Der Standard umfasst daher nicht nur das Kommunikationsprotokoll, sondern auch das Datenmodell des Servers. OPC UA wird für die zukünftige industrielle Kommunikation eine große Bedeutung beigemessen [75].

Der Hauptanwendungsfall von OPC UA für eine SPS wird zwar darin gesehen, dass die SPS als Server auftritt. Die SPS kann aber auch die Rolle des Klienten einnehmen und selbst Daten von anderen Systemen abfragen. Es wurden daher auch IEC 61131-3 -kompatible Funktionsbausteine von der OPC Foundation veröffentlicht [62]. Die Bausteine sind speziell auf die Verwendung des OPC UA-Protokolls und -Datenmodells ausgerichtet und lassen sich nicht auf andere Kommunikationstechnologien übertragen.

IEC 61131-5

In der Norm IEC 61131-5 bzw. DIN EN 61131-5 [17] wird auf die Schnittstelle zu anderen Systemen aus einer allgemeineren Sicht und ohne Annahmen über das konkrete Kommunikationssystem eingegangen. Wegen des breiteren Anwendungsbereichs wird an dieser Stelle auf die Norm genauer eingegangen.

Der Anwendungsbereich der Norm umfasst die SPS selbst und deren Schnittstellen, aber nicht die Kommunikationspartner. Wenn eine SPS die Rolle eines Klienten einnimmt, beispielsweise bei einer Datenabfrage, wird deshalb davon ausgegangen, dass

sich der Server als Kommunikationspartner ebenfalls wie eine SPS verhält. Sonst würde sein Verhalten außerhalb des Anwendungsbereichs der Norm liegen.

In der DIN EN 61131-5 wird eine Liste von Kommunikationsfunktionen definiert. Dies sind: Gerätestatus lesen, Datenlesen, Steuern, Synchronisation zwischen Anwendungen, Melden, Programmausführung und E/A-Steuerung, Transfer des Anwendungsprogramms, Verbindungsmanagement. Im Kontext dieser Arbeit ist die Funktion „Datenlesen“ relevant. Für diese Funktion kann eine SPS sowohl als Server als auch als Klient auftreten.

In der Norm wird für das Datenlesen zwischen zwei Anwendungsfällen unterschieden: Das Datenlesen als Datenanforderung und das Datenlesen als Datenbereitstellung. Im ersten Fall bestimmt die lesende SPS, wann Daten angefordert werden. Die SPS sendet dann eine Datenabfrage an das betreffende Fremdgerät und erhält zu einem späteren, nicht kontrollierbaren Zeitpunkt eine Antwort. Im zweiten Fall wird der Kommunikationspartner so programmiert, dass er die geforderten Daten selbst beim Auftreten eines bestimmten Ereignisses oder unter bestimmten Bedingungen absendet. Insofern ist der zweite Fall, auch wenn er in der Norm als Fall von „Datenlesen“ bezeichnet wird, eher ein konfiguriertes Datenschreiben.

Für die praktische Implementierung der Kommunikationsfunktionen werden in der Norm Funktionsbausteine spezifiziert. Es werden die Schnittstellen und das Verhalten der Bausteine angegeben. Hier ist insbesondere der Funktionsbaustein READ interessant, weil durch ihn das Datenlesen als Datenabfrage realisiert wird.

Schnittstellen des Funktionsbausteins READ

Der Zweck des Bausteins READ ist es, den Wert bzw. die Werte von einer oder mehreren Variablen abzufragen, die beim Kommunikationspartner gespeichert sind. Dazu besitzt der Baustein einen „ID“ genannten Eingang, dessen Wert einen Kommunikationskanal identifiziert. In der Terminologie der IEC 61131-5 hat dieser Eingang daher den Datentyp COMM_CHANNEL. Seitens der Norm wird nicht weiter spezifiziert, wie der Kommunikationskanal technisch implementiert ist. Es wird nur vorausgesetzt, dass durch den Wert von ID genau ein Kommunikationskanal zu genau einem Kommunikationspartner identifiziert wird und dass dieser Kommunikationskanal zur Übertragung diskreter Nachrichten geeignet ist. Über weitere technologieabhängige Aspekte, die im praktischen Einzelfall relevant sein können (beispielsweise Datendurchsatzrate, Latenz, Sicherheit, Verlässlichkeit), wird keine Aussage getroffen.

Neben dem Eingang ID hat der READ-Baustein die Eingänge VAR_1,...,VAR_N ($N \geq 1$). Durch diese Eingänge wird jeweils eine Variable identifiziert. In vielen Fällen passiert das durch die Angabe eines Variablenpfads. Der Datentyp dieser Eingänge ist entsprechend STRING oder VAR_ADDR (ein spezieller Datentyp zur Adressierung von Variblen). Grundsätzlich muss es dem Kommunikationspartner aber nur möglich sein, anhand der Werte von VAR_1,...,VAR_N Variablen zu identifizieren.

Schließlich besitzt der READ-Baustein noch den Eingang „REQ“ vom Datentyp BOOL, durch den bei einer steigenden Flanke das Lesen des Wertes ausgelöst wird. Nach erfolgreichem Lesen wird für einen Taktzyklus der BOOL-Ausgang „NDR“ (new data received) auf den Wert true gesetzt. Für den Fall eines Fehlers zeigen die Ausgänge ERROR (BOOL) und STATUS (INT) an, dass ein Fehler aufgetreten ist und um welche Art von Fehler es sich handelt. Beispielsweise bedeutet der Wert 0 „kein Fehler“, der Wert

6 „Empfänger nicht bereit“ und der Wert 1 „Fehler in unterer Kommunikationsschicht“. Herstellerspezifische Werte für den Status sind ebenfalls möglich.

Die empfangenen Werte werden in den Variablen RD_1,..., RD_N gespeichert. Diese Variablen sind als VAR_IN_OUT deklariert, damit sie von dem Programm, das die Daten verwendet auch beschreibbar sind. Der Datentyp ist ANY. Die vollständige Spezifikation der Variablen des Bausteins READ entsprechend DIN EN 61131-5 [17] wird in Codelisting 3.2 gezeigt.

```
VAR_INPUT
        REQ     : BOOL R_EDGE;  (* Datenlesen                          *)
        ID      : COMM_CHANNEL; (* Kommunikationskanal                 *)
        VAR_1   : STRING;       (* Werte der Bezeichner der angeforderten *)
          :                     (* Variablen erweiterbar,              *)
        VAR_N : STRING;         (* Typ VAR_ADDR auch moeglich          *)
END_VAR

VAR_OUTPUT
        NDR     : BOOL;         (* Neue Anwenderdaten empfangen        *)
        ERROR : BOOL;           (* Neuer Status nicht 0 empfangen      *)
        STATUS: INT;            (* Zuletzt festgestellter Status       *)
END_VAR

VAR_IN_OUT
        RD_1    : ANY;          (* Empfangene Anwenderdaten,           *)
          :                     (* erweiterbar und von                 *)
        RD_N    : ANY;          (* beliebigem Datentyp                 *)
END_VAR
```

Listing 3.2: ST-Quellcode zur Spezifikation der Variablen des Funktionsbausteins READ.

Verhalten des Funktionsbausteins READ

Das Laufzeitverhalten des READ-Bausteins wird in der DIN EN 61131-5 durch einen Zustandsautomaten spezifiziert. Abbildung 3.4 zeigt diese Logik informell als Diagramm in Ablaufsprache (Sequential Function Chart). Nach der Initialisierung befindet sich ein READ-Baustein zunächst im Leerlauf, bis am Eingang REQ eine steigende Flanke anliegt. Dann wird – auf nicht näher spezifizierte Art – eine Leseanfrage an den Kommunikationspartner geschickt, der durch COMM_CHANNEL identifiziert wird. Das Ziel dieser Anfrage ist durch die Werte von VAR_1 bis VAR_N gegeben. Nach Erhalt einer Antwort wird der Inhalt der Antwort überprüft. Das betrifft insbesondere die Datentypen der Daten in der Antwort, die zu RD_1 bis RD_N kompatibel sein müssen. Nach erfolgreicher Prüfung werden die Daten in RD_1 bis RD_N geschrieben und der Funktionsbaustein geht zurück in den Leerlaufzustand.

Bei Ausführung des Ablaufs ist an zwei Stellen die Behandlung von Fehlern vorgesehen. Erstens kann die Kommunikation mit dem Kommunikationspartner fehlschlagen, was am Kommunikationskanal liegen kann oder daran, dass der Kommunikationspartner die Leseoperation nicht ausführen kann. Zweitens können die erhaltenen Daten feh-

lerhaft sein, z.B. falsche Datentypen enthalten oder unvollständige Daten. Die Prüfroutinen hierfür werden durch die Norm aber nicht definiert. In beiden Fehlerfällen wird der Fehler durch den Ausgang ERROR für einen Taktzyklus angezeigt und der Wert von Status dem Fehler entsprechend gesetzt. Der Baustein geht zurück in den Leerlaufzustand.

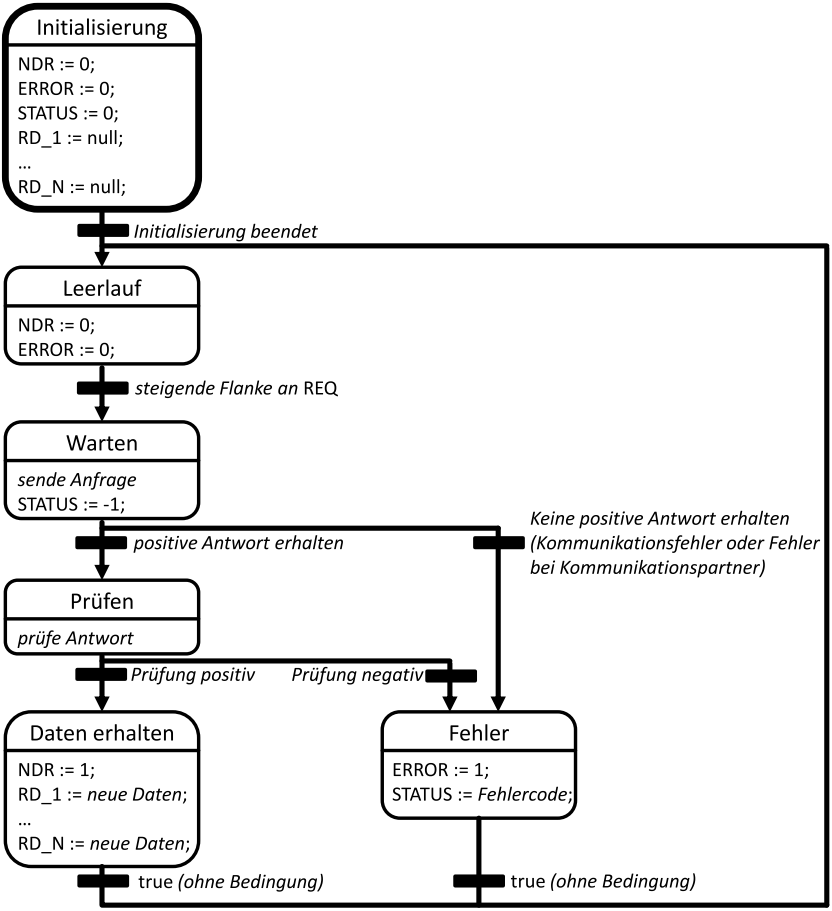

Abbildung 3.4.: Ablauflogik des Funktionsbausteins READ entsprechend DIN EN 61131-5 [17], dargestellt in Ablaufsprache (Sequential Function Chart).

4. Analyse und Anforderungen

In Abschnitt 1.3 wurden wichtige Leitfragen dieser Arbeit gestellt. Mithilfe der Grundlagen aus Kapitel 2 und dem Wissen über den Stand der Technik aus Kapitel 3 werden diese Fragen nun beantwortet. Daraus werden Anforderungen an eine Lösung abgeleitet.

4.1. Eignung der Merkmalmodelle

Aus wissenschaftlicher Sicht ist die Frage danach, wie technische Merkmale modelliert werden, fast ausschließlich im Hinblick auf die technische Abbildung und Verwendung der Merkmale untersucht worden. Es gibt viele Arbeiten, die beispielsweise vorhandene Katalogsysteme analysieren oder die die Möglichkeiten spezieller Technologien zur Informationsrepräsentation (z.B. Ontologien) zeigen und untersuchen. Das in Abschnitt 2.1 beschriebene (Meta-) Modell für die Modellierung von Merkmalen ist eines der wenigen, evtl. sogar das einzige in der Literatur vorhandene Modell, das explizit nur die Klärung der Modellierung von Merkmalen zum Ziel hat. Es beantwortet gerade nicht die Frage, welche Merkmale es gibt und wie sie in Geschäfts- und technischen Prozessen verwendet werden, wohl aber welche Information für ein vollständiges Modell verfügbar sein muss. Für den Anwender wird durch dieses Modell klar, auf welche Struktur sich die Abfrage einer Merkmalinformation bezieht. Das Merkmalmodell aus Abschnitt 2.1 ist somit das semantische Ziel der zu definierenden Abfragesprache.

Anforderung 1: *Der Abfragesprache muss ausschließlich das in Abschnitt 2.1 definierte Merkmalmodell für die Semantik von Abfragen zugrunde liegen.*

Aus praktischer Sicht ist die Frage nach tatsächlich im industriellen Umfeld genutzten Datenmodellen wichtiger als die Modelle aus der Literatur. Schließlich ist die beste Abfragesprache nutzlos, wenn sie nicht auf Daten zugreifen kann. Daher wurden in Abschnitt 3.1 die praktisch vorhandenen und genutzten Modelle erläutert. Dabei ergibt sich ein gemischtes Bild. Positiv ist, dass bereits seit vielen Jahren technische Merkmale systematisch definiert und verwaltet werden und dass Katalogsysteme auf diese Merkmale Bezug nehmen. Beispiele dafür sind eCl@ss und UNSPSC. Viele Organisationen haben ihre Spezifikationen auch untereinander ausgetauscht und abgeglichen, so dass umfangreiche und einheitliche Kataloge entstanden sind.

Positiv ist ebenfalls, dass es für die technische Speicherung und den Austausch von Merkmalen Normen gibt und dass diese Normen in den letzen Jahren zum großen Teil miteinander harmonisiert wurden. Dadurch hat die IEC 61360 eine wichtige zentrale Bedeutung erhalten.

Der negative Aspekt der aktuellen Situation ist, dass diese vorhandenen Normen und Standards für die Abwicklung von Geschäftsprozessen zwischen Unternehmen konzi-

piert sind. Für andere Anwendungsfälle, in denen Informationen über Merkmale Gegenstand der Kommunikation sind, werden die Standards nicht angewendet. Zum Teil ist das auch logisch nicht möglich, z.B. weil eCl@ss für die Beschreibung von Produkttypen konzipiert ist und nicht für die Modellierung einzelner Produkte (d.h. es behandelt nur Merkmalträgertypen). Insgesamt bedeutet das, dass standardisierte Datenformate für Merkmale eine mögliche Datenquelle sind, die aber nicht in jedem Fall vorhanden, nutzbar und ausreichend sind.

Im industriellen Umfeld gibt es zahlreiche Software-Systeme, die prinzipiell für eine SPS in der Produktion nützliche Information zu Merkmalen enthalten. Die wichtigsten davon wurden in Abschnitt 3.3 beschrieben. Durch Normen wie die ISA-95 gibt es auch klare Vorstellungen darüber, welche Informationen von diesen Systemen verwaltet werden. Mögliche Lösungen für offene Datenschnittstellen sind durch B2MML und ein ISA-95 -konformes OPC UA -Profil vorhanden, nach aktuellem Stand wird aber keine Standard-Datenschnittstelle von Softwareherstellern durchgängig unterstützt. Der Zugriff auf die umfangreiche Information über technische Merkmale, die in den Modellen der ISA-95 vorhanden sind, ist daher im Allgemeinen nicht auf Ebene der Modelle möglich. Stattdessen muss auf die darunter liegenden Informationssysteme zugegriffen werden, d.h. üblicherweise relationale Datenbanken oder konventionelle Dateisysteme. Für speziellere Datenrepräsentationen wie XML, eCl@ss, B2MML oder OPC UA müssen entsprechende Adapter implementiert werden. Die zu definierende Abfragesprache bildet dabei keine Ausnahme.

Anforderung 2: *Die Abfragesprache muss den Zugriff auf Informationen ermöglichen, die in üblichen Informationssystemen gespeichert sind. Dazu gehören insbesondere Dateisysteme und relationale Datenbanken.*

4.2. Entwurf der Abfragesprache

Wegen der unterschiedlichen möglichen Datenquellen muss sich die Abfragesprache nach dem Anwender und nicht nach dem Datenanbieter richten. Entsprechend Anforderung 1 ist die logische Sicht des Anwenders auf Merkmale klar. Über das Vorgehen zum Erlangen von Information wird dadurch aber noch keine Aussage gemacht. Gerade wegen der heterogenen Kommunikationspartner ist es für den Anwender wünschenswert, wenn er keine Details über die Kommunikation mit dem jeweiligen merkmalverwaltenden System haben muss. Andersherum betrachtet gewährleistet diese Abschirmung des Anwenders den notwendigen Freiheitsgrad für Veränderungen der Datenquellen ohne Rückwirkungen. Die Abfrage von Information sollte daher als Benutzung eines Dienstes im Sinne von Abschnitt 2.1.2 bzw. DIN SPEC 40912 [24] funktionieren.

Anforderung 3: *Die Verwendung der Abfragesprache muss der Verwendung eines Dienstes zur Abfrage von Merkmalinformation entsprechen.*

Wichtig an Anforderung 3 ist, dass die Verwendung der Abfragesprache der Verwendung eines Dienstes nur entspricht. Wie genau der Dienstleister arbeitet, wo er realisiert wird und unter wessen Kontrolle er sich befindet, wird nicht festgelegt. Entsprechend

Anforderung 2 können ganz unterschiedliche Datenquellen genutzt werden und es ist durchaus möglich, dass die gefragte Information aus einer lokalen Datei stammt, über die der Anwender selbst volle Kontrolle hat.

Entsprechend Anforderung 1 soll der Dienst Anfragen betreffend das Merkmalmodell aus Abschnitt 2.1 verarbeiten können. Abschnitt 2.1.2 gibt einen Überblick über die zu realisierenden Basisoperationen für merkmalverwaltende Systeme. Darüber hinaus werden aber auch zusätzliche Operationen benötigt, die komplexere Abfragen zulassen. Wie am Ende von Abschnitt 2.1.2 beschrieben, ist es praktisch oft nicht ausreichend, wenn nur die Basisoperationen realisiert werden. Stattdessen müssen auch komplexe Abfragen möglich sein, in denen Informationen bedingt abgefragt werden oder Abfragen durch Verknüpfung hierarchisch aufgebaut werden können.

Anforderung 4: *Die Abfragesprache muss die Formulierung komplexer Abfragen ermöglichen, so dass Informationen bedingt abgefragt werden können und hierarchisch aufgebaute Abfragen möglich sind.*

Anforderung 4 ist absichtlich offen in Bezug auf die tatsächlich zu implementierenden Operationen gehalten, weil in dieser Arbeit keine vollständige und abschließende Auflistung der Operationen erfolgen soll. Komplexe Operationen, die in bestimmten Anwendungsfällen häufig auftreten, sollten als neue Operatoren formulierbar sein, um die Anwendung zu vereinfachen. Hier soll nur eine Grundmenge von Operationen formuliert werden, die für möglichst viele Anwendungsfälle ausreicht.

Zur eindeutigen Spezifikation der Abfragesprache wird eine formale Grundlage benötigt. Dadurch kann zugesichert werden, dass eine konkrete Abfrage eine eindeutige Semantik hat. Wäre das nicht der Fall, dann könnte das Ergebnis einer Abfrage vom Speicherort der Information abhängen, weil ggf. unterschiedliche Implementierungen verwendet werden – z.B. ein Algorithmus für das Durchsuchen einer Datei und ein anderer Algorithmus für die Abfrage an einer Datenbank. Daher gilt Anforderung 5:

Anforderung 5: *Der Spezifikation der Abfragesprache muss eine formale Sprache mit klar definierter Semantik zugrunde liegen.*

Außerdem sollte die Sprache in möglichst vielen Anwendungsfällen tatsächlich nutzbar sein. Die Zahl der Anwendungsfälle ist aber nicht begrenzt, deshalb kann von praktischer Seite die Abdeckung der Fälle nicht geprüft werde. Es ist daher hilfreich, wenn klare Aussagen über grundsätzlich mögliche, d.h formulierbare, und nicht mögliche Abfragen gemacht werden können.

Anforderung 6: *Die Ausdruckstärke der Sprache muss dokumentiert sein.*

In Abschnitt 2.2 wurden das relationale Datenbankmodell und die damit verbundene relationale Algebra vorgestellt. Das relationale Datenbankmodell hat eine formale Basis und es ist eine umfangreiche Theorie vorhanden. Die relationale Algebra ist abgeschlossen und lässt dadurch auch komplexe hierarchische Abfragen zu. Das bedeutet andererseits, dass komplexe Abfragen in atomare Basisoperationen zerlegt werden können, was die Implementierung vereinfacht und strukturiert. Außerdem kann durch Verwendung relationaler Algebra klare Aussagen über Möglichkeiten und Grenzen der Abfragespra-

che getroffen werden, beispielsweise durch das Zeigen relationaler Vollständigkeit.

In der Praxis werden heute auch häufig graphbasierte Datenbanken und Abfragesprachen angetroffen (siehe Abschnitt 3.2). Diese besitzen ebenfalls eine gut erforschte formale Grundlage, können Ihre praktischen Vorteile aber nur in Anwendungen mit sehr stark strukturierten Daten ausspielen, was im Fall des hier verwendeten Merkmalmodells nicht gegeben ist. Insgesamt ist daher das relationale Datenbankmodell eine sinnvolle Grundlage für die Formulierung des Datenmodells und der Datenabfragen.

Zur Wahl des relationalen Datenbankmodells muss angemerkt werden, dass das relationale Datenbankmodell nur für die formale Spezifikation der Abfragesprache verwendet werden soll. Ob die Daten tatsächlich in einer relationalen Datenbank abgelegt sind, ist davon vollkommen unabhängig. Außerdem ist das zugrunde gelegte relationale Informationsmodell für Merkmale für den Benutzer der Abfragesprache nicht direkt sichtbar. Es ist nur eine Hilfe bei der Erstellung der Sprache, durch die die Anwendbarkeit der Abfragesprache zugesichert wird.

4.3. Implementierung

Wie in Abschnitt 3.4 beschrieben, ist die IEC 61131 (bzw. DIN EN 61131 [18]) der dominierende Standard für die Programmierung von SPS. Wenn die Abfragesprache von einer SPS aus nutzbar sein soll, ist die Konformität mit diesem Standard daher der einzige sinnvolle Weg. Wie beschrieben gibt es laut der Norm fünf Programmiersprachen, von denen sich FBS als eine universell einsetzbare Sprache etabliert hat. FBS ist auch deswegen besonders unter den Programmiersprachen, weil sie gleichermaßen in Problemstellungen mit einer kontinuierlichen und einer zeitdiskreten Berechnung der Lösung eingesetzt werden kann. Hilfreich ist auch, dass FBS eine graphische Programmiersprache ist, die leichter zugänglich ist als eine textuelle Sprache mit eigener Syntax. Insgesamt verspricht die Implementierung von Funktionsbausteinen entsprechend der IEC 61131-3 in der Form, dass mit den Funktionsbausteinen Abfragen formuliert werden können, die höchstmögliche Akzeptenz und Anwendbarkeit.

Anforderung 7: *Abfragen müssen durch IEC 61131-3 -konforme Funktionsbausteine formulierbar und ausführbar sein.*

Wie die interne Logik der Funktionsbausteine implementiert ist, wird durch Anforderung 7 nicht ausgesagt.

Die Implementierung als FBS bringt eine inhärente Eigenschaft mit sich: Abfragen werden deklarativ formuliert, denn FBS ist eine deklarative Programmiersprache. Der Anwender der Sprache muss also nicht das prozedurale Vorgehen zur Berechnung des Ergebnisses einer Abfrage formulieren, sondern muss nur die Eigenschaften dieses Ergebnisses formulieren. Wenn z.B. ein Merkmalträger mit einer bestimmten Aussage über ein Merkmal gesucht wird, dann muss der Anwender nur diese Aussage angeben und nicht die Suchprozedur. Das ist offensichtlich komfortabler und entspricht auch dem Paradigma anderer Datenabfragesprachen wie SQL.

Von theoretischer Seite her ist das deklarative Paradigma von FBS unproblematisch. Wie in Abschnitt 3.2 festgestellt wurde, ist der Relationenkalkül ebenfalls deklarativ und gegenüber relationaler Algebra gleich mächtig. Wenn die Semantik eines Funktionsbau-

steins im Relationenkalkül formuliert ist, kann sie also auch in relationale Algebra übersetzt werden. Für die Implementierung der internen Logik des Funktionsbausteins, die ggf. in einer imperativen Programmiersprache geschieht, ist diese Darstellung dann geeigneter und leichter umsetzbar.

In Abschnitt 3.4.3 wurden Möglichkeiten gezeigt, um mit Mitteln der IEC 61131 Datenabfragen zu realisieren. Eine generische Möglichkeit ohne die Bindung an eine bestimmte Kommunikationstechnologie ist durch die IEC 61131-5 [17] gegeben. Dort ist der Bausteintyp READ spezifiziert, der gerade für Datenabfragen von Fremdsystemen bestimmt ist. Eine Abfrage in der zu definierenden Abfragesprache unterscheidet sich im Wesentlichen darin von der Funktion des READ-Bausteins, dass der genaue Speicherort der gefragten Information für den Anwender unbekannt ist (und nicht als Variablenpfad bekannt). Aufgrund der bestehenden Ähnlichkeit der Aufgabenstellung gilt:

Anforderung 8: *Ausführungslogik und Schnittstellen des Bausteins zur Ausführung einer Abfrage müssen so weit wie möglich mit dem READ-Baustein der IEC 61131-5 übereinstimmen.*

Wie in Abschnitt 3.4.2 diskutiert wurde, ist das Verhalten eines Funktionsbausteinnetzwerks von der Ausführungsreihenfolge der Bausteine abhängig. Hier bedeutet das, dass das Ergebnis einer Abfrage beeinflusst werden kann. Dieser Einfluss muss für den Anwender klar nachvollziehbar sein.

Anforderung 9: *Die Bedeutung der Ausführungsreihenfolge von Funktionsbausteinen der Abfragesprache muss dokumentiert und leicht nachvollziehbar sein.*

Aus Anforderung 3, laut der eine Anwenderschnittstelle entsprechend einem Dienst verlangt wird, ergibt sich eine weitere Anforderung für die Implementierung. Bei einem Dienst liegt die Arbeitslast beim Dienstleister, während der Klient in der Zeit zwischen dem Aufruf einer Operation und dem Ende der Ausführung bzw. Erhalt des Resultats selbst keine Arbeitslast trägt. Darüber hinaus ist es für den Klienten meist auch unerheblich, welcher Arbeitsaufwand durch die Ausführung einer Operation hervorgerufen wird. Gerade im Anwendungsbereich einer SPS ist das eine besonders vorteilhafte Eigenschaft eines Dienstes, weil hier oft Echtzeitbedingungen eingehalten werden müssen. Wenn die SPS selbst eine deutlich höhere oder stark schwankende Arbeitslast für die Ausführung von Abfragen zu tragen hat, kann das zum Ausschlusskriterium für die Anwendung der Abfragesprache werden. Aus diesem Grund wurde in Abschnitt 2.1.2 darauf hingewiesen, dass der Dienstleister möglichst auch komplexere Operationen auf Merkmaldaten zur Verfügung stellen sollte. Allgemein muss daher Anforderung 10 erfüllt werden:

Anforderung 10: *Die Ausführung einer Abfrage darf den Betrieb des abfragenden Systems nicht beeinflussen.*

Mögliche Aufwände zum Erstellen der Abfrage und Verarbeitung des Ergebnisses werden durch Anforderung 10 jedoch nicht berücksichtigt. Außerdem ist es möglich,

dass durch eine zu lange Zeitdauer bis zum Erhalt des Ergebnisses die Funktionalität beeinflusst wird. Auch hierüber wird durch Anforderung 10 keine Aussage gemacht.

SPS-basierte Automatisierungssysteme werden oft über sehr lange Zeiträume (Jahre oder sogar Jahrzehnte) betrieben. In diesen Zeiträumen sind Veränderungen der umgebenden IT-Infrastruktur sehr wahrscheinlich und auch merkmalverwaltende Systeme können betroffen sein. In vielen Anwendungsfällen der Automatisierungstechnik ist es aber nur mit erheblichem Aufwand oder auch überhaupt nicht möglich, den Betrieb zu unterbrechen. Das bedeutet, dass es trotz Veränderungen der IT-Infrastruktur (d.h. der Datenanbindung) möglich sein muss, die Abfragen von Merkmalinformation durchgehend zu verwenden. Daraus folgt:

Anforderung 11: *Die Implementierung muss Änderungen der Konfiguration der Datenanbindung von merkmalverwaltenden Systemen ohne Unterbrechung des Betriebs unterstützen.*

Für einen möglichst hohen Nutzwert ist es wünschenswert, wenn Information aus vielen und unterschiedlichen Datenquellen abgerufen werden kann. Die Erweiterbarkeit in Hinblick auf verwendbare Datenquellen muss daher in der Implementierung besondere Beachtung finden.

Anforderung 12: *Die Implementierung muss die Anbindung zusätzlicher Datenquellen in einfacher und dokumentierter Art unterstützen.*

5. Lösungskonzept

In diesem Kapitel wird die Abfragesprache für Merkmalinformation definiert und eine Systemarchitektur festgelegt. Der Aufbau dieses Kapitels und die Zusammenhänge zu den vorausgegangenen Kapiteln werden in Abbildung 5.1 gezeigt. In Abschnitt 5.1 wird dazu das Merkmalmodell aus Abschnitt 2.1 auf das relationale Datenbankmodell übertragen. Dadurch gibt es eine formale Grundlage für die Strukturen, aus denen Daten abgefragt werden.

Im darauf folgenden Abschnitt 5.2 wird eine Menge von Grundoperationen für das formalisierte Merkmalmodell definiert. Der Hauptzweck dieser Grundoperationen ist es, eine Grundlage für speziellere und komplexere Operationen zu bilden, die aus den Grundoperationen zusammengesetzt werden können. Daher sind die Grundoperationen nicht durch praktische Anwendungsszenarien motiviert, sondern so gestaltet, dass sie den Aufbau möglichst vieler und mächtiger zusammengesetzter Operationen zulassen. So muss später nur eine möglichst kleine Anzahl von Operationen auf unterster Ebene implementiert werden, ohne dass die Anwendbarkeit darunter leidet.

In Abschnitt 5.3 werden dann die Operationen definiert, die aus praktischer Sicht nutzenbringend sind.

Die Verwendung der zuvor definierten Operationen in Funktionsbausteinen (entsprechend Anforderung 7 und 8) wird in Abschnitt 5.4 beschrieben. Damit sind Syntax und Semantik der Abfragesprache definiert.

Die Ausführung der Funktionsbausteine ist mit gewissen Voraussetzungen an das technische Umfeld verknüpft (s. Anforderung 3, 10 und 11). Deshalb wird in Abschnitt 5.5 eine Systemarchitektur entsprechend den gestellten Anforderungen beschrieben.

5.1. Abbildung des Merkmalmodells im relationalen Datenbankmodell

5.1.1. Formale Spezifikation

Die Abbildung auf das relationale Datenbankmodell wird durch die Definition von Relationsschemas durchgeführt. Jedes operativ nutzbare Element des Merkmalmodells entsprechend Abschnitt 2.1 entspricht einem Relationsschema. Für die Attribute dieser Schemas gibt es drei semantische Kategorien:

- Das Attribut *Id* identifiziert ein Tupel innerhalb seines Anwendungsbereichs (z.B. Betrieb oder Standort) eindeutig und ist daher ein Primärschlüssel (mit Ausnahme des Merkmalträgers, s.u.). Zur Herstellung der Eindeutigkeit kann ein Bezeichner der höheren Ebene, z.B. des Datenbanksystems oder eine URL, vorangestellt werden. Der Wert von *Id* wird innerhalb eines Ausdrucks der Abfragesprache nicht als Zeichenkette, sondern wie das identifizierte Tupel selbst behandelt.

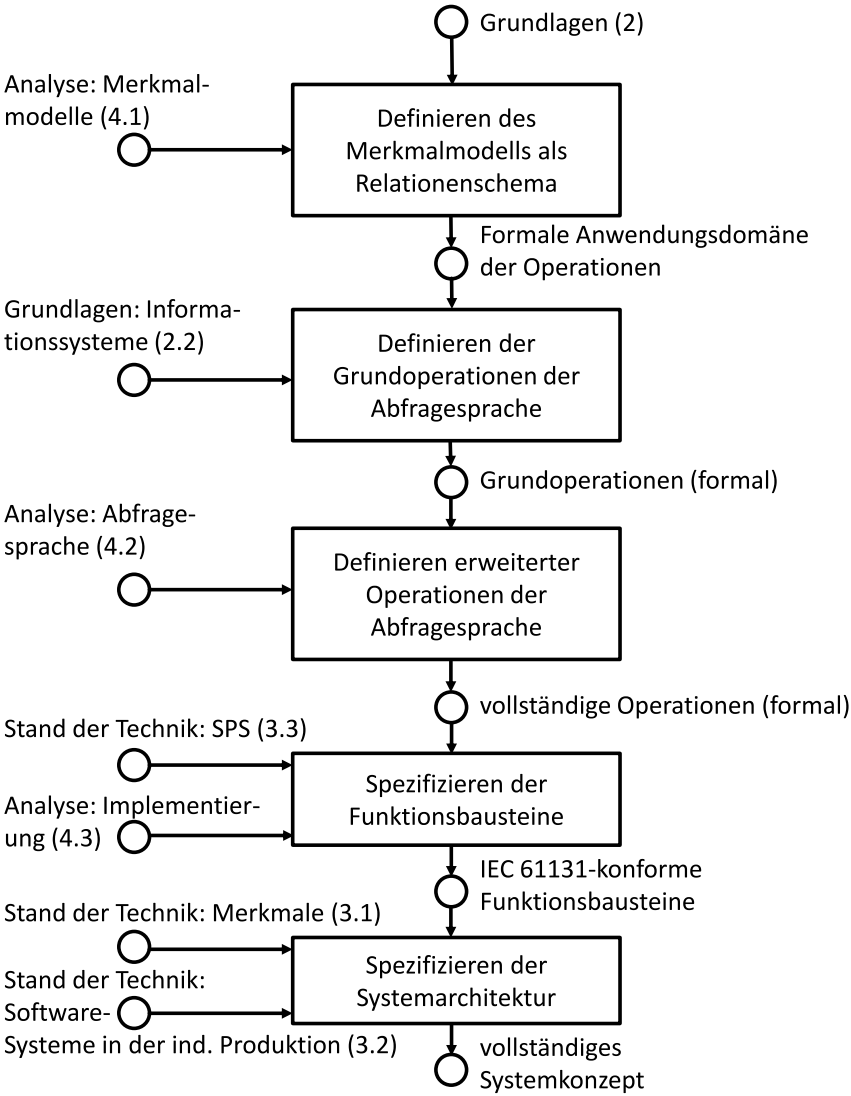

Abbildung 5.1.: Aufbau von Kapitel 5.

- Die Attribute *Wert*, *Relation* und *Einheit* vermitteln Zusatzinformation über den Wert, mit dem ein Merkmal durch eine Aussage assoziiert wird.

- Alle übrigen Attribute bilden die Relationen zwischen den Elementen des Modells ab und dienen zur Navigation zwischen Tupeln mittels *Id*-Attributen.

An dieser Stelle werden nur die zur operativen Verwendung notwendigen Attribute definiert. Es kann zwar davon ausgegangen werden, dass Elemente wie Merkmale und Merkmalträgertypen natürlichsprachliche Namen und Definitionen besitzen und dass in vielen Fällen noch weitere Information verfügbar ist. Diese ist aber meist nicht operativ nutzbar, der Name würde beispielsweise von der lokalen Sprache abhängen oder kann ambivalent sein. Außerdem soll hier die Anwendbarkeit nicht durch zu starke Annahmen über verfügbare Information begrenzt werden.

Die Definitionen von Merkmalträgertyp und Merkmalträger sehen es hier nicht vor, dass entsprechende Tupel selbst auf Aussagen verweisen. Stattdessen werden die Relationsschemas so definiert, dass Aussagen auf die Merkmalträgertypen bzw. Merkmalträger verweisen, auf die sie sich beziehen. Für die hier ausschlaggebende theoretische Sicht ist auch nicht relevant, in welche Richtung Verweise existieren, weil Aussagen und das betroffene Aussageziel durch die Vereinigungsoperation $\bowtie$ zusammengeführt werden können. In einer Implementierung kann mit beliebigen einseitigen oder doppelten Verweisen gearbeitet werden.

Die folgenden Absätze definieren Relationsschemas zur Repräsentation des Merkmalmodells. Zur Verdeutlichung werden die Relationsschemas durch ein tiefgestelltes S gekennzeichnet.

Merkmalträgertyp

$$MTT_S = \left\{ Id : \Sigma^*, AllgMerkmale : \{\Sigma^*\}^n, Supertypen : \{\Sigma^*\}^m, Aussagen : \{\Sigma^*\}^k \right\}$$

Id: Zeichenkette, identifiziert den Merkmalträgertyp eindeutig.

AllgMerkmale: $0 < n$-stelliger Vektor von Zeichenketten, in dem jeder Eintrag durch einen Wert $AM_S.Id$ auf ein allgemeines Merkmal verweist, das dem Merkmalträgertyp zugeordnet wird. Alle Einträge des Vektors sind paarweise verschieden. [Bemerkung: Zur Vereinfachung wird angenommen, dass der Wertebereichstyp „Vektor von Zeichenketten“ verfügbar ist. Er ließe sich sonst durch eine Hilfskonstruktion nachbilden.]

Supertypen: $0 \leq m$-stelliger Vektor von Zeichenketten, in dem jeder Eintrag durch einen Wert $MTT_S.Id$ auf einen Merkmalträgertyp verweist, von dem geerbt wird. Die Einträge sind paarweise verschieden. Die induzierte Vererbungsstruktur zwischen Merkmalträgertypen ist hierarchisch, also insbesondere kreisfrei.

Aussagen: $0 \leq k$-stelliger Vektor von Zeichenketten, in dem jeder Eintrag auf durch einen Wert $AS_S.Id$ auf eine Aussage verweist, die für jeden Merkmalträger des Merkmalträgertyps gilt.

Allgemeines Merkmal

$$AM_S = \{Id : \Sigma^*, Merkmalart : \Sigma^*\}$$

Id: Zeichenkette, identifiziert das allgemeine Merkmal eindeutig.

Merkmalart: Zeichenkette mit Wert $MA_S.Id$, verweist auf die zugehörige Merkmalart.

Merkmalart

$$MA_S = \{Id : \Sigma^*, Supertyp : \Sigma^*\}$$

Id: Zeichenkette, identifiziert die Merkmalart eindeutig.

Supertyp: Zeichenkette mit Wert $MA_S.Id$, identifiziert eine andere Merkmalart oder einen Merkmalprototyp, von dem geerbt wird. Die induzierte Vererbungsstruktur zwischen Merkmalarten ist kreisfrei.

Merkmalprotyp

$$MP_S = \{Id : \Sigma^*\}$$

Id: Zeichenkette, identifiziert den Merkmalprototyp eindeutig.

Aussageart

$$AA_S = \{Id : \Sigma^*\}$$

Id: Zeichenkette, identifiziert die Aussageart eindeutig.

Merkmalträger

$$MT_S = \{Id : \Sigma^*, Merkmalträgertyp : \Sigma^*\}$$

Id: Zeichenkette, identifiziert den Merkmalträger eindeutig.

Merkmalträgertyp: Zeichenkette mit Wert $MTT_S.Id$, verweist auf den zugehörigen Merkmalträgertyp.

Aussage

$$AS_S = \{Id : \Sigma^*, Merkmalträger : \Sigma^*, AllgMerkmal : \Sigma^*, Aussageart : \Sigma^*, \\ Relation : \{=, <, \leq, >, \geq, \approx\}, Einheit : \Sigma^*, Wert : \mathbb{X}\}$$

Id: Zeichenkette, identifiziert die Aussage eindeutig.

Merkmalträger: Zeichenkette mit Wert $MT_S.Id$ oder $MTT_S.Id$, die den Merkmalträger oder Merkmalträgertyp identifiziert, auf den sich die Aussage bezieht.

AllgMerkmal: Zeichenkette mit Wert $AM_S.Id$, die das allgemeine Merkmal identifiziert, über das eine Aussage getroffen wird. [Bemerkung 1: Wenn die Aussage einen Merkmalträger betrifft, dann identifiziert *AllgMerkmal* ein spezielles Merkmal des Merkmalträgers. Trotzdem wird als Kennung die eines allgemeinen Merkmals verwendet, weil durch die Kombination mit *Merkmalträger* klar wird, dass ein spezielles Merkmal gemeint ist, ohne dass hierfür neue Kennungen eingeführt werden müssen. Bemerkung 2: Das allgemeine Merkmal muss dem Merkmalträgertyp bzw. dem Merkmalträgertyp des Merkmalträgers zugeordnet sein.]

Aussageart: Zeichenkette mit Wert $AA_S.Id$, die die Aussageart identifiziert, die die Semantik der Aussage definiert.

Relation: Bestimmt, wie das Merkmal, über das eine Aussage getroffen wird, zu einem Wert in Relation gesetzt wird.

Wert: Maschinell verarbeitbarer Wert, zu dem das Merkmal, über das eine Aussage getroffen wird, in Relation gesetzt wird. [Bemerkung: $\mathbb{X}$ entspricht der Menge aller Werte des Datentyps „ANY“ der IEC 61131.]

Einheit: Zeichenkette, durch die die Einheit von *Wert* definiert wird (z.B. als SI-Einheit).

5.1.2. Anwendersicht

Für den Anwender, der letztendlich Funktionsbausteine zur Abfrage von Information verwenden möchte, ist das Denken in Relationsschemas umständlich und erscheint unpassend. Die in Abschnitt 2.1 gewählte Darstellung in Form von Klassendiagrammen ist im Vergleich anschaulicher und besser geeignet, um das Grundmodell zu verstehen. Es muss aber auch berücksichtigt werden, dass ein Anwender eine Lösung für ein konkretes Problem sucht und daher eher an dem Aussehen konkreter Daten interessiert ist, d.h. der „Instanzebene“, als am Grundkonzept der Modellierung auf der Klassenebene. Eine einfache und effektive Hilfestellung ist deshalb das Aufführen von Beispielen für Merkmaldaten in Form von Tabellen.

In Tabelle 5.1 und den darauf folgenden Tabellen werden Beispieldaten für jedes Relationsschema gezeigt. Zu den Merkmalträgern A0815 und A0816, beide vom Typ „Kreiselpumpe mit Wellendichtung“ entsprechend eCl@ss 9.0, werden Aussagen über die maximale Förderhöhe und Förderstrom gemacht. Aussagen und Merkmalträger werden der Übersichtlichkeit halber direkt in einer Tabelle zusammengeführt. Die Definitionen der allgemeinen Merkmale und Merkmalarten sind durch eCl@ss gegeben. Merkmalprototypen als „Supertypen“ von Merkmalarten sind in eCl@ss nicht vorhanden, daher wird auf die ISO 80000 [16] verwiesen. Die in der ISO 80000 referenzierten Größen entsprechen den jeweils in eCl@ss verwendeten Einheiten. Für die Aussageart wird die interne Konvention verwendet, dass „Z01“ eine Zusicherung des Herstellers ist, z.B. aus einem Datenblatt. Die Beispieldaten können dazu verwendet werden, um die korrekte

Funktion der Pumpen A0815 und A0816 zu überprüfen.

Tabelle 5.1.: Beispieldaten entsprechend einer Kombination der Relationsschemas MT_S und AS_S.

Merkmalträger $\bowtie_{MT_S.Id=AS_S.Merkmalträger}$ Aussage						
Id	**Merkmalträgertyp**	**AllgMerkmal**	**Aussageart**	**Relation**	**Einheit**	**Wert**
A0815	ecl@ss9.0/36-41-01-08	ecl@ss9.0/-0173-1#02-BAJ123#006	Z01	$\geq$	m	127.0
A0815	ecl@ss9.0/36-41-01-08	ecl@ss9.0/-0173-1#02-BAI023#003	Z01	$\geq$	m^3/h	110.0
A0816	ecl@ss9.0/36-41-01-08	ecl@ss9.0/-0173-1#02-BAJ123#006	Z01	$\geq$	m	105.0
A0816	ecl@ss9.0/36-41-01-08	ecl@ss9.0/-0173-1#02-BAI023#003	Z01	$\geq$	m^3/h	122.0

5.2. Grundoperationen der Abfragesprache

Zunächst sollen Grundoperationen festgelegt werden, durch die später auch komplexere Operationen definiert werden können. Dazu werden die Grundoperationen so gewählt, dass sie per Konstruktion relational vollständig sind. So kann von Beginn an zugesichert werden, dass zur Abdeckung der Ausdrucksstärke der relationalen Algebra keine weiteren Operationen benötigt werden.

5.2.1. Abdeckung der relationalen Algebra

Die hier verwendeten Grundoperationen sind Projektion, Selektion, Kreuzprodukt, Vereinigung, Differenz und Umbenennung. Gegenüber den ursprünglichen acht Operationen der relationalen Algebra fehlen bei dieser Auswahl die Operationen Division, Restriktion, Schnittmenge und Vereinigung, d.h. zum Zeigen der relationalen Vollständigkeit müssen die fehlenden vier Operationen auf die ausgewählten sechs Grundoperationen zurückgeführt werden. Für die Division und die Restriktion wurde das bereits von Codd bei der Definition der Operationen gezeigt [10]. Die Schnittmengenoperation $\cap$ kann auf die Differenzoperation zurückgeführt werden, weil für die Menge X und Y gilt $X \cap Y = X \setminus \{X \setminus Y\}$. Die Vereinigungsoperation kann durch das Kreuzprodukt und die Selektion gebildet werden (s. Seite 21). Somit ist die genannte Menge von Grundoperationen relational vollständig.

Die Operationen können nun mit Einschränkung auf die Relationsschemas aus Abschnitt 5.1 definiert werden, weil sie ohnehin nur in diesem Kontext verwendet werden. Bei dieser Gelegenheit wird auch die Notation als Funktionen eingeführt, die einfacher

Tabelle 5.2.: Beispieldaten entsprechend dem Schema MTT_S (mit zusätzlicher Bemerkung).

Merkmalträgertypen			
Id	**AllgMerkmale**	**Supertypen**	Bemerkung
ecl@ss9.0/36-41-01-08	ecl@ss9.0/-0173-1#02-BAJ123#006; ecl@ss9.0/-0173-1#02-BAI023#003	ecl@ss9.0/36-41-01	Kreiselpumpe mit Wellendichtung
ecl@ss9.0/36-41-01		ecl@ss9.0/36-41	Kreiselpumpe

Tabelle 5.3.: Beispieldaten entsprechend dem Schema AM_S (mit zusätzlicher Bemerkung).

Allgemeine Merkmale		
Id	**Merkmalart**	Bemerkung
ecl@ss9.0/0173-1#02-BAJ123#006	ecl@ss9.0/02-BAJ123	Max. Förderhöhe
ecl@ss9.0/0173-1#02-BAI023#003	ecl@ss9.0/02-BAI023	Max. Förderstrom

Tabelle 5.4.: Beispieldaten entsprechend dem Schema MA_S (mit zusätzlicher Bemerkung).

Merkmalarten		
Id	**Supertyp**	Bemerkung
ecl@ss9.0/02-BAJ123	ISO80000-3-1.1	Max. Förderhöhe
ecl@ss9.0/02-BAI023	ISO80000-4-30	Max. Förderstrom

Tabelle 5.5.: Beispieldaten entsprechend dem Schema MP_S (mit zusätzlicher Bemerkung).

Merkmalprototypen	
Id	Bemerkung
ISO80000-3-1.1	Länge
ISO80000-4-30	Volumenstrom

Tabelle 5.6.: Beispieldaten entsprechend dem Schema AA_S (mit zusätzlicher Bemerkung).

Aussagearten	
Id	Bemerkung
Z01	Zusicherung des Herstellers

lesbar ist. Diese Funktionen werden dann im Kapitel 5.4 von den Funktionsbausteinen verwendet.

Es sei $\mathbb{A}$ die Menge aller Attribute aus Abschnitt 5.1, die in den Relationsschemas verwendet werden, $\mathbb{T}$ die Menge aller Tupel entsprechend der hier verwendeten Relationsschemas, $\mathbb{BF}$ die Menge aller booleschen Formeln und $\mathfrak{P}$ das Symbol für die Potenzmenge. Es gelte
$\{\} \neq T \subset \mathbb{T}$, $\{\} \neq U \subset \mathbb{T}$, $\{\} \neq A \subset \mathbb{A}$, $t \in \mathbb{T}$, $a, \tilde{a} \in \mathbb{A}$ und $b \in \mathbb{BF}$. Dann sind die Grundoperationen definiert durch:

Projektion

$$reduce : \mathfrak{P}(\mathbb{T}) \times \mathfrak{P}(\mathbb{A}) \to \mathfrak{P}(\mathbb{T})$$

$$reduce(T, A) = \pi_A(T)$$

Selektion

$$choose : \mathfrak{P}(\mathbb{T}) \times \mathbb{BF} \to \mathfrak{P}(\mathbb{T})$$

$$choose(T, b) = \sigma_b(T)$$

Kreuzprodukt

$$combine : \mathfrak{P}(\mathbb{T}) \times \mathfrak{P}(\mathbb{T}) \to \mathfrak{P}(\mathbb{T})$$

$$combine(T, U) = T \times U$$

Vereinigung

$$union : \mathfrak{P}(\mathbb{T}) \times \mathfrak{P}(\mathbb{T}) \to \mathfrak{P}(\mathbb{T})$$

$$union(T, U) = T \cup U$$

Differenz

$$remove : \mathfrak{P}(\mathbb{T}) \times \mathfrak{P}(\mathbb{T}) \to \mathfrak{P}(\mathbb{T})$$

$$remove(T, U) = T \setminus U$$

Umbenennung

$$rename : \mathfrak{P}(\mathbb{T}) \times \mathbb{A} \times \mathbb{A} \to \mathfrak{P}(\mathbb{T})$$

$$rename(T, a, \tilde{a}) = \rho_{\tilde{a}/a}(T)$$

5.2.2. Wertausgabe

Diese so festgelegten Operationen sind noch nicht operativ nutzbar. Ein Grund dafür ist, dass alle Funktionen in eine Menge von Tupeln abbilden. Der Anwender der Abfragesprache möchte und kann aber nicht mit Tupeln als Antworten auf Abfragen arbeiten, sondern mit Werten der Menge $\mathbb{X}$, die von den IEC 61131-3 -Programmiersprachen weiterverarbeitet werden können. Dafür wird zusätzlich die Funktion *value* für die Rückgabe eines Wertes bzw. mehrerer Werte benötigt.

$$value : \mathfrak{P}(\mathbb{T}) \times \mathbb{A} \rightarrow \mathbb{X}^n$$

$$value(T, a) = t_1.a, ..., t_n.a, \qquad T = \{t_1, ..., t_n\}$$

5.2.3. Boolesche Formeln

Letztlich ist noch die Verwendung boolescher Formeln $\mathbb{BF}$ zu klären, die in der Selektion verwendet werden. Die Programmiersprachen der IEC 61131 verwenden boolesche Ausdrücke, so dass hier kein neues Konzept eingeführt werden muss und auf die vorhandene Syntax zurückgegriffen werden kann. Die in den Formeln verwendeten Variablen sind in diesem Fall aber keine Variablen der SPS, sondern Attributwerte. Wenn aus den Aussagen T beispielsweise diejenigen herausgesucht werden sollen, für die das Attribut *Aussageart* den Wert „Zusicherung“ hat und *Relation* „$\geq$“ ist, kann dazu die Funktion

$$choose(T, Aussageart = \text{Zusicherung} \wedge Relation = \geq)$$

verwendet werden.

5.2.4. Vererbungsbeziehungen

Eine der bekannten Grenzen der relationalen Algebra ist das Fehlen einer transitiven Hülle. Im Anwendungskontext der Schemas aus Abschnitt 5.1 macht sich das durch zwei Einschränkungen bemerkbar:

1. Mit den bisher definierten Grundoperationen können die „Supertypen“ eines Merkmalträgertyps nicht iteriert werden. Z.B. könnte die Frage „Ist dieser Merkmalträger vom Typ Produkt?“ nicht beantwortet werden, weil die vollständige Vererbungshierarchie nicht durchsucht oder aufgelistet werden kann. Das gilt prinzipiell auch für die Vererbungsrichtung „nach unten“.

2. Analog dazu ist es nicht möglich, die übergeordneten (oder untergeordneten) Merkmalarten einer Merkmalart aufzulisten oder zu durchsuchen. Das kann dann erforderlich sein, wenn entschieden werden soll, ob ein Gegenstand mit unbekanntem Merkmalträgertyp aufgrund seines Gesamtgewichts von einer Maschine, z.B. einem Transportsystem, verarbeitet werden kann. In dem Fall ist das konkrete allgemeine Merkmal, das geprüft werden muss, unbekannt, weil das Gesamtgewicht bei jedem Merkmalträgertyp unterschiedlich definiert ist. Es ist aber bekannt, dass es von „Gesamtgewicht“ abgeleitet sein muss.

Zur Überwindung dieser Einschränkungen wird die rekursive Funktion $typeOf$ für Merkmalarten (MA_S) und Merkmalträgertypen (MTT_S) definiert, die eine gesamte Vererbungshierarchie zurückgibt. Die Funktion deckt nur den Anwendungsfall der Hierarchie „nach oben“ ab, weil es für den Fall „nach unten“ keine praktisch sinnvolle Motivation gibt.

$$typeOf : \mathfrak{P}(\mathbb{T}) \times \mathfrak{P}(\mathbb{T}) \to \mathfrak{P}(\mathbb{T})$$

$$typeOf(T, U) = \begin{cases} \bigcup_{u \in U} choose(T, Id = u.Supertyp) & \\ \quad \cup\, typeOf(T, value(u, Supertyp)), & \text{alle } u \text{ entsprechen } MA_S \\ \bigcup_{u \in U} choose(T, \bigvee_i Id = u.Supertypen_i) & \\ \quad \bigcup typeOf(T, value(T, Supertypen)_i), & \text{alle } u \text{ entsprechen } MTT_S \\ \{\}, & \text{sonst} \end{cases}$$

5.2.5. Aggregationen

Eine weitere wesentliche Einschränkung der relationalen Algebra ist, dass sie keine Operationen zur Aggregation von Werten enthält. Wenn beispielsweise der Merkmalträger gefunden werden soll, bei dem ein bestimmtes Merkmal die maximale Ausprägung unter allen Merkmalträgern hat, etwa die Bestellung mit der höchsten Priorität, dann müssen mit den bisher definierten Funktionen alle Merkmalträger abgerufen und überprüft werden. Auch wenn das kein direkter Konflikt mit Anforderung 10 ist, nach der die Ausführung der Abfrage keinen Einfluss auf den Betrieb der SPS haben soll, so widerspricht es trotzdem der Intention dieser Anforderung. Schließlich wächst der Aufwand mit der Menge an Daten, die durchsucht werden müssen, und diese Menge ist der SPS vorab nicht bekannt. Insofern ist es sinnvoll, die vorhandenen Funktionen um Möglichkeiten zur Aggregation von Zahlenwerten zu erweitern.

Bei der Aggregation von Werten gibt es zwei unterschiedliche Wege der Aggregation: Die Aggregation über mehrere Merkmalträger hinweg (z.B. Finden des schwersten, größten oder ältesten Merkmalträgers) und die Aggregation von Werten innerhalb desselben Merkmalträgers (z.B. Bildung von Differenzen zwischen oberem und unterem Grenzwert, Berechnung zusammengesetzter Merkmale wie Dichte, ...). Für diese Fälle werden zwei Funktionen definiert: $aggrAll$ und $aggrEach$. Wünschenswert ist, dass die Funktionen schachtelbar sind, damit beispielsweise bei gegebener Dichte und Volumen für jeden Merkmalträger derjenige mit der höchsten Masse gefunden werden kann. Dafür muss zunächst die Darstellung eines Ergebnisses durch das Relationsschema

$$E_S = \{Id : \Sigma^*, Wert : \mathbb{X}\}$$

definiert werden. Die Verwendung eines eigenen Schemas für Ergebnisse hat einen einfachen Hintergrund: So ist es möglich, Ergebnisse einer Berechnung als Argument einer anderen Funktion zu verwenden. Die Geschlossenheit der Funktionen wird also nicht aufgebrochen. Für die Id, die ein Ergebnis trägt, wird die Id des Merkmalträgers verwendet, der dieses Ergebnis erzeugt hat. Durch die Benennung des Attributs $Wert$, die bei AS_S und E_S gleich ist, braucht in der Funktion $aggrAll$ nicht zwischen Aussage und Ergebnis als Argument unterschieden zu werden.

Sei nun $\mathbb{E}$ die Menge aller Ergebnisse entsprechend dem Schema E_S, $\mathbb{MT}$ die Menge aller Merkmalträger entsprechend dem Schema MT_S und $\mathbb{AS}$ die Menge aller Aussagen entsprechend AS_S. $\odot$ sei das Symbol für einen Operator aus der Menge $\{+, -, *, /, max, min, \#\}$. Dann kann die Funktion *aggrAll* durch eine Fallunterscheidung von $\odot$ definiert werden. Im Fall der Operatoren $+, -, *, /$ werden die *Wert*-Attribute der an die Funktion übergebenen Tupel durch den entsprechenden Operator (den Wert von $\odot$) verknüpft. Für *max* und *min* wird das Maximum bzw. Minimum der *Wert*-Attribute gesucht. Der Operator # zählt die übergebenen Tupel. Es gilt also:

$$aggrAll : \mathfrak{P}(\mathbb{E} \cup \mathbb{AS}) \times \{+, -, *, /, max, min\} \rightarrow \mathbb{E}$$

$$aggrAll(T, \odot) = \begin{cases} \{e | e \in \mathbb{E} \wedge e.Wert = t_1^* \odot ... \odot t_n^* \wedge e.Id = t_1.Id\}, \\ \qquad\qquad \text{wenn } \odot \in \{+, -, *, /\} \\ \{e | e \in \mathbb{E} \wedge e.Wert = \odot\{t.Wert | t \in T\} \\ \quad \wedge e.Id = \{t.Id | t.Wert = e.Wert\}\}, \qquad \text{wenn } \odot \in \{max, min\} \\ \{e | e \in \mathbb{E} \wedge e.Wert = |T| \wedge e.Id = t_1.Id\}, \qquad \text{wenn } \odot \in \{\#\} \end{cases}$$

$$\text{mit } T^* = \{t_1^*, ..., t_n^*\} = \{t_1.Wert, ..., t_n.Wert\} \subset \{\mathbb{E} \cup \mathbb{AS}\}.$$

Mit der Funktion *aggrEach* werden beliebige Attribute innerhalb eines Tupels mittels der Operatoren $+, -, *, /$ verknüpft. Wenn mehrere Tupel an die Funktion übergeben werden, geschieht das für jedes Tupel. Wenn mehrere Operatoren übergeben werden, geschieht die Anwendung in der übergebenen Reihenfolge (d.h. die Priorisierung „Punkt vor Strich“ existiert nicht).

$$aggrEach : \mathfrak{P}(\mathbb{T}) \times \mathbb{A}^n \times \{+, -, *, /\}^{n-1} \rightarrow \mathfrak{P}(\mathbb{E})$$

$$aggrEach(T, (a_1, ..., a_n), (\odot_1, ..., \odot_{n-1})) = \\ \{e | e \in \mathbb{E} \wedge e.Wert = t_i.a_1 \odot_1 ... \odot_{n-1} t_i.a_n \wedge e.Id = t_i.Id \wedge i \in \{1, ..., |T|\}\}$$

$$n \in \{1, 2, ...\}, T = \{t_1, ..., t_k\} \subset \mathbb{T}, \odot_i \in \{+, -, *, /\}$$

Die Anwendung der Funktion zur Aggregation selbst ist verhältnismäßig einfach, kann aber je nach Aufgabenstellung in einem komplizierten Kontext geschehen. Als kurzes Beispiel zur Illustration soll der oben genannte Fall realisiert werden, in dem der Merkmalträger mit der größten Masse gesucht wird. Gegeben sind Aussagen über die Istwerte von Dichte und Volumen zu jedem Merkmalträger in $MT \subset \mathbb{MT}$ in jeweils derselben Einheit.

Im ersten Schritt werden die Aussagen aus $AS \subset \mathbb{AS}$ ausgewählt, die Merkmalträger aus MT betreffen. Dazu werden die Merkmalträger und Aussagen durch die Funktionen

choose und *combine* passend zusammengeführt. Weil sowohl Aussagen als auch Merkmalträger das Attribut *Id* besitzen, wird dieses zur Vermeidung einer Namenskollision für die Aussagen in $Id2$ umbenannt. Die neue Menge von Tupeln aus Merkmalträgern und Aussagen ist dann MT^*.

$$MT^* = choose(combine(MT, rename(AS, Id, Id2)), Id = Merkmalträger))$$

Im nächsten Schritt werden die Tupel MT^*_{Dichte} und $MT^*_{Volumen}$ ausgewählt, die Informationen zu den genauen Istwerten der Dichte bzw. des Volumens beinhalten.

$$MT^*_{Dichte} = choose(MT^*, AllgMerkmal = \text{Dichte} \wedge Aussageart = \text{Istwert} \wedge Relation = =)$$

$$MT^*_{Volumen} = choose(MT^*, AllgMerkmal = \text{Volumen} \wedge Aussageart = \text{Istwert} \wedge Relation = =)$$

Anschließend werden einige Attribute umbenannt, um danach die Tupel ohne Namenskollision zusammenführen zu können.

$$\tilde{MT}^*_{Volumen} = rename(rename(MT^*_{Volumen}, Id, Id2), Wert, Volumentwert)$$

$$\tilde{MT}^*_{Dichte} = rename(MT^*_{Dichte}, Wert, Dichtewert)$$

Die Tupel werden nun über die gemeinsame *Id* verknüpft.

$$MT^*_{VolumenDichte} = choose(combine(\tilde{MT}^*_{Dichte}, \tilde{MT}^*_{Volumen}), Id = Id2)$$

Anschließend folgt die Berechnung der Massen je Tupel.

$$ErgebnisseMasse = aggrEach(MT^*_{VolumenDichte}, (Volumenwert, Dichtewert), (*))$$

Daraus kann schließlich das Maximum ermittelt und der Wert mittels *value* ausgegeben werden.

$$MaxMasse = aggrAll(ErgebnisseMasse, max)$$

$$x = value(Wert, MaxMasse)$$

Das Beispiel lässt erkennen, dass komplexe Abfragen durch die definierten Funktionen möglich sind, dass diese aber auch kleinteilig und umständlich sein können. Im folgenden Abschnitt werden deshalb spezielle Operationen für in der Praxis häufig auftretende Aufgaben definiert.

5.3. Erweiterte Operationen der Abfragesprache

Aufbauend auf den zuvor definierten Grundoperationen werden nun erweiterte Operationen definiert. Die Auswahl ist durch praktische Anwendungsbeispiele motiviert. Für die Definition wird nur auf die vorhandenen Grundoperationen und andere Operationen dieses Abschnitts zurückgegriffen. Die hier genannten erweiterten Operationen sind nur als Beispiele zu verstehen und können durch weitere Operationen ergänzt werden.

5.3.1. Zusammenführen von Merkmalträgern und Aussagen

Beschreibung

Es ist häufig notwendig, Merkmalträger und Aussagen, die diese betreffen, zusammenzuführen. In Tabelle 5.1 (Seite 59) ist ein einfaches Beispiel dazu zu sehen. Zusätzlich können auch Aussagen existieren, die den Merkmalträgertyp zum Ziel haben und dadurch indirekt für einzelne Merkmalträger gelten. Als Vereinfachung für das Zusammenführen der Information wird dafür eine eigene Operation definiert.

Anwendungsbeispiel

Zu einer Auswahl von Merkmalträgern sollen alle relevanten Aussagen gesucht werden.

Definition

$$joinStatements : \mathfrak{P}(\mathbb{T}) \times \mathfrak{P}(\mathbb{MT}) \rightarrow \mathfrak{P}(\mathbb{T})$$

$$joinStatements(T, MT) = union(\{directStatementInformation\}, \{indirectStatementInformation\})$$

Zur Berechnung müssen die direkten Aussagen über den Merkmalträger und die indirekten Aussagen, die durch den Merkmalträgertyp gemacht werden, ermittelt werden. Dabei gilt:

$T \subset \mathbb{T}$, $MT \subset \mathbb{MT}$, MTT sei die Teilmenge von T entsprechend MTT_S und AS sei die Teilmenge von T entsprechend AS_S,

$$\{directStatementInformation\} = reduce(\{rawDirectStatementInformation\}, \{Id, Merkmalträgertyp, AllgMerkmal, Aussageart, Relation, Einheit, Wert\}),$$

$$\{rawDirectStatementInformation\} = choose(combine(MT, rename(AS, Id, Id2)), Id = Merkmalträger),$$

$$\{indirectStatementInformation\} = (\{rawIndirectStatementInformation\}, \{Id, Merkmalträgertyp, AllgMerkmal, Aussageart, Relation, Einheit, Wert\}),$$

$$\{rawIndirectStatementInformation\} = (combine(\{propertyCarrierWithAllSupertypes\}, rename(AS, Id, Id2)), supertypeId = Merkmalträger),$$

$$\{propertyCarrierWithAllSupertypes\} = reduce(choose(combine(MT, rename(join(MTT, typeOf(T, MTT)), Id, supertypeId)), supertypeId = Merkmalträgertyp), \{Id, supertypeId, Merkmalträgertyp\}).$$

5.3.2. Suche nach Merkmalträgern

Beschreibung

Die Operation ermöglicht das Suchen nach Merkmalträgern mit bestimmten Eigenschaften. Als gesuchte Eigenschaften können der Merkmalträgertyp *mtt*, die Aussageart *aa* einer Aussage über den Merkmalträger, die Relation *rel* der Aussage, das betreffende Merkmal *mm*, der Aussagewert *we* und Einheit *eh* wahlweise eingesetzt werden. Dabei dürfen einzelne Variablen unbelegt bleiben und werden in dem Fall ignoriert. Durchsucht wird die Menge T von Daten.

Anwendungsbeispiel

Suche alle Merkmalträger vom Typ „Bestellung“ aus den Daten T, bei denen das Merkmal Auftragsnummer mit dem Wert „x“ übereinstimmt.

Definition

$$whichones : \mathfrak{P}(\mathbb{T}) \times \mathfrak{P}(\mathbb{MT}) \times \mathbb{T} \times \mathbb{T} \times \mathbb{T} \times \{=, <, \leq, >, \geq\} \times \mathbb{X} \times \Sigma^* \to \mathfrak{P}(\mathbb{T})$$

$$whichones(T, MT, mtt, aa, mm, rel, we, eh) = choose(joinStatements(T, MT), Merkmalträgertyp \stackrel{*}{=} mtt \wedge Aussageart \stackrel{*}{=} aa \wedge AllgMerkmal \stackrel{*}{=} mm \wedge Relation \stackrel{*}{=} rel \wedge Wert \stackrel{*}{=} we \wedge Einheit \stackrel{*}{=} eh)$$

Mit $(b \stackrel{*}{=} \tilde{b}) \Leftrightarrow (b = \tilde{b})$, wenn $b \neq \varepsilon \wedge \tilde{b} \neq \varepsilon$, *true* sonst.

$b, \tilde{b} \in \mathbb{BF}$, ε als Symbol für eine nicht belegte Variable.

5.3.3. Aggregationen

Beschreibung

In Abschnitt 5.2 wurden bereits die Operationen *aggrEach* und *aggrAll* zur Durchführung von Aggregationen definiert. Aus praktischer Sicht ist es sinnvoll diese Operationen so zu erweitern, dass als Parameter auch Merkmalträger und Merkmalträgertypen (zusammen mit Aussagen über diese) übergeben werden können. Die Aggregation wird dann auf bestimmte Aussagen über die übergebenen Merkmalträger(typen) angewendet, so dass diese Aussagen nicht durch eine vorherige Operation gesucht werden müssen. Dadurch wird die Anwendung in vielen Fällen erheblich vereinfacht. Die erweiterten Operationen erhalten die Namen $aggrEach'$ und $aggrAll'$.

Anwendungsbeispiel $aggrEach'$

Für eine Untermenge von Merkmalträgern aus einem Datensatz soll die Masse anhand von Aussagen über Dichte und Volumen ermittelt werden (vgl. Beispiel auf Seite 64).

Definition von $aggrEach'$

$$aggrEach' : \mathfrak{P}(\mathbb{T}) \times \mathfrak{P}(\mathbb{MT}) \times \mathbb{A}^n \times \{+,-,*,/\}^{n-1} \times \mathbb{T} \times \mathbb{T} \times \mathbb{T} \times \mathbb{T} \times \mathbb{T} \times \mathbb{T} \to \mathfrak{P}(\mathbb{E})$$

$$aggrEach'(T, MT, (a_1,...,a_n), (\odot_1,...,\odot_{n-1}), mtt, aa, mm, rel, we, eh) = \begin{cases} aggrEach(T, (a_1,...,a_n), (\odot_1,...,\odot_{n-1}))), & \text{wenn } MT = \{\} \\ aggrEach(whichones(T, MT, mtt, aa, mm, rel, we, eh), (a_1,...,a_n), (\odot_1,...,\odot_{n-1})), & \text{sonst} \end{cases}$$

$$n \in \{1,2,...\}, \odot_i \in \{+,-,*,/\}$$

Anwendungsbeispiel $aggrAll'$

Für eine Untermenge von Merkmalträgern aus einem Datensatz soll derjenige mit dem größten Ist-Wert der Masse herausgesucht werden (vgl. Beispiel auf Seite 64).

Definition $aggrAll'$

$$aggrAll' : \mathfrak{P}(\mathbb{T}) \times \{+,-,*,/,max,min\} \times \mathfrak{P}(\mathbb{MT}) \times \mathbb{T} \times \mathbb{T} \times \mathbb{T} \times \mathbb{T} \times \mathbb{T} \times \mathbb{T} \to \mathbb{E}$$

$$aggrAll'(T, MT, \odot, mtt, aa, mm, rel, we, eh) = \begin{cases} aggrAll(T, \odot), & \text{wenn } MT = \{\} \\ aggrAll(reduce(whichones(& \\ \quad T, MT, mtt, aa, mm, rel, we, eh), & \\ \quad \{\ Id, Wert\}), \odot), & \text{sonst} \end{cases}$$

5.3.4. Bestimmung des Merkmalträgertyps

Beschreibung

Die Bestimmung des Merkmalträgertyps eines Merkmalträgers kann direkt durch das Abfragen des entsprechenden Attributs geschehen. Wenn die Fragestellung aber lautet, ob ein Merkmalträger m aus T entweder von einem bestimmten Typ typ ist, oder von einem Typ, der von diesem Typ erbt, kann die Funktion $isoftype$ verwendet werden.

Anwendungsbeispiel

In einem automatischen Lagersystem ist der exakte Typ der eingelagerten Gegenstände nicht relevant. Es muss aber grundsätzlich zwischen unterschiedlichen Klassen unterschieden werden.

Definition

$$isoftype : \mathfrak{P}(\mathbb{T}) \times \mathbb{MT} \times \mathbb{T} \rightarrow \mathbb{T}$$

$$isoftype(T, m, typ) = choose(typeOf(T, value(m, Merkmalträgertyp)), Id = typ)$$

5.3.5. Vorhandensein eines Merkmals

Beschreibung

Die Operation bestimmt, ob es für den Merkmalträger m Aussagen über ein Merkmal der Merkmalart art gibt oder über eine Merkmal von einer Merkmalart, die von art erbt. Falls dies der Fall ist, wird die Menge der zutreffenden Merkmalarten zurückgegeben, sonst die leere Menge.

Anwendungsbeispiel

Die Weiterverarbeitung eines hängt davon ab, ob Informationen über bestimmte Qualitätsmerkmale vorhanden sind.

Definition

$$hasaproperty : \mathfrak{P}(\mathbb{T}) \times \mathbb{MT} \times \mathbb{T} \rightarrow \mathbb{T}$$

Die Funktion wird in mehreren Schritten definiert:

$$hasaproperty(T, m, art) = chose(\{AllGenericProperties\}, Id = art)$$

Mit $AllGenericProperties$ als Menge aller allgemeinen Merkmale, über die es direkt oder indirekt Aussagen gibt:

$$\{AllGenericProperties\} = typeof(T, \{GenericProperties\})$$

GenericProperites ist die Menge der von durch Aussagen betroffenen allgemeinen Merkmalen:

$$\{GenericProperties\} = chose(T, value(T, Id) = value(\{Properties\}, Merkmalart))$$

$$\{Properties\} = chose(T, value(joinStatements(T, m), AllgMerkmal) = Id)$$

5.3.6. Verknüpfung von Merkmalträgern

Beschreibung

Die Operation gibt aus zwei Mengen von Merkmalträgern diejenigen zurück, die in ausgewählten Attributen übereinstimmen bzw. bei denen die Aussagewerte in einer bestimmten Relation stehen. Ob die Attribute in der gewählten Relation stehen müssen, wird durch Boolesche Wahrheitswerte für jedes Attribut vorgegeben.

Anwendungsbeispiel

Posten einer Bestellung werden mit vorhandenen Lagerbeständen abgeglichen.

Definition

$$matching: \mathfrak{P}(\mathbb{T}) \times \mathfrak{P}(\mathbb{MT}) \times \mathfrak{P}(\mathbb{MT}) \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \{=, <, \leq, >, \geq, \{\}\} \to \mathfrak{P}(\mathbb{MT})$$

$$\begin{aligned} &matching(T, MT_1, MT_2, b_{Id}, b_{Merkmalträgertyp}, \\ &\qquad b_{AllgMerkmal}, b_{Aussageart}, b_{Relation}, b_{Einheit}, rel_{Wert}) \\ &\quad = reduce(choose(joinStatements(T, \{MT_1, MT_2\}), b^*), \{Id, Merkmalträgertyp\}) \end{aligned}$$

mit

$$\begin{aligned} b^* = &(Id = Id2 \vee \neg b_{Id}) \\ &\wedge (Merkmalträgertyp = Merkmalträgertyp2 \vee \neg b_{Merkmalträgertyp}) \\ &\wedge (AllgMerkmal = AllgMerkmal2 \vee \neg b_{AllgMerkmal}) \\ &\wedge (Aussageart = Aussageart2 \vee \neg b_{Aussageart}) \\ &\wedge (Relation = Relation2 \vee \neg b_{Relation}) \\ &\wedge (Einheit = Einheit2 \vee \neg b_{Einheit}) \\ &\wedge (Wert \overset{*}{rel}_{Wert} Wert2) \end{aligned}$$

mit $Wert \overset{*}{rel}_{Wert} Wert2 = Wert\ rel_{Wert}\ Wert2$, wenn $rel_{Wert} \in \{=, <, \leq, >, \geq\}$, *true* sonst.

5.4. Schnittstellen und Verhalten der Funktionsbausteine

In diesem Kapitel werden aus den Operationen, die in den Abschnitten 5.2 und 5.3 definiert wurden, Funktionsbausteine abgeleitet. Dazu gehört die Spezifikation der Schnittstellen der Bausteine gemäß IEC 61131-3 und die Spezifikation der SPS-seitigen Ausführung einer Abfrage. Nicht dazu gehört die Spezifikation der Algorithmen, die die zuvor definierten Operationen implementieren. Gemäß Anforderung 3 soll die Ausführung einer Abfrage der Verwendung eines Dienstes entsprechen, daher liegen diese Algorithmen außerhalb der Zuständigkeit der SPS und sind technisch davon abhängig, wie die Merkmalinformation gespeichert wird.

5.4.1. Konzept

In den vorigen Kapiteln wurde durch einige Beispiele gezeigt, dass die Operationen der Abfragesprache sich ineinander schachteln lassen, um so komplexe Abfragen aufzubauen. Solche komplexen Abfragen sollten, genau wie einfache Abfragen aus einer einzigen Operation, seitens der SPS als eine einzige Abfrage behandelt werden. Die Interpretation und Ausführung solcher komplexer Abfragen obliegt dann ganz dem (Fremd-)System, das als Dienstleister zur Ausführung der Abfragen auftritt. Auf diese Art wird die SPS in ihrer Funktion nicht dadurch beeinflusst, wie komplex die Ausführung einer Abfrage ist (siehe Anforderung 10).

Damit besteht die Bearbeitung einer Abfrage seitens der SPS aus zwei Phasen, nämlich der Generierung einer Abfrage als geschachtelter Ausdruck und anschließend die eigentliche Durchführung der Abfrage. Die Generierung ist deshalb eine eigene Phase, weil die Abfrage durch Variablen veränderliche Inhalte enthalten kann. Zum Aufbau von Ausdrücken werden die zuvor definierten Operationen verwendet: Wenn es zu jeder Operationen einen entsprechenden Funktionsbausteintyp gibt, dann bildet der Funktionsbaustein lediglich eine Zeichenkette, die den konkret zu realisierenden Aufruf als Text wiedergibt. Diese Zeichenkette wird als Wert in den Ausgang des Funktionsbausteins geschrieben und kann als Argument für einen anderen Funktionsbaustein verwendet werden. Ein Beispiel dazu zeigt Abbildung 5.2. Darin wird eine Operation durch eine geschachtelte Abfrage aus *value*, *typeOf* und *choose* aufgebaut, die prüft, ob ein Merkmalträger vom Merkmalträgertyp ProduktZ ist und die ggf. diesen Typ zurückgibt. Die Bausteintypen sind für das Beispiel willkürlich festgelegt. Wir nehmen im Beispiel an, dass der Wert von MerkmalträgerX einen Merkmalträger anhand seiner *Id* identifiziert und dass der Wert von DatenbankY eine Datenbank (oder auch mehrere Datenbanken) identifiziert. Am Ausgang jedes Funktionsbausteins ist die Zeichenkette angegeben, die durch den Baustein aufgebaut wurde. Letztendlich wird der Variable qXisTypeZ die Abfrage als Wert zugewiesen. Eine äquivalente Abfrage könnte auch durch eine andere Reihenfolge der Bausteine erreicht werden. qXisTypeZ beinhaltet am Ende die ausformulierte gesamte Abfrage, die an dieser Stelle aber noch nicht ausgeführt ist. Alternativ könnte eine entsprechende Zeichenkette auch durch vorhandene String-Operationen aufgebaut werden. Ein Anwender kann die Funktionsbausteine oder die Repräsentation als Zeichenkette verwenden, ohne dass ihm die jeweils andere Methode bekannt sein muss.

Nach dem Aufbau einer Abfrage wird die Abfrage durch einen eigenen Funktionsbaustein durchgeführt. Dazu benötigt ein Baustein mindestens drei Eingänge: Einen für

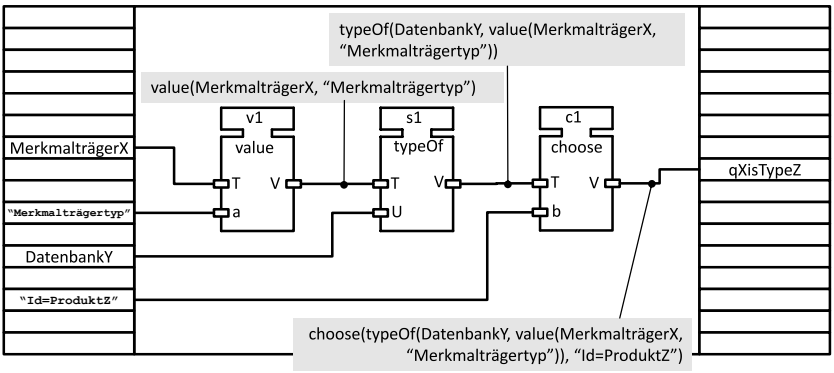

Abbildung 5.2.: Beispiel für den Aufbau einer Abfrage aus Funktionsbausteinen.

die zuvor aufgebaute Abfrage, einen Eingang, an dem durch eine steigende Flanke das Signal zur Ausführung der Abfrage gegeben wird, und letztlich einen Eingang, an dem das gefragte Attribut angegeben wird. Das Attribut ist notwendig, weil die Ergebnisse von Abfragen Tupel sind, die von einer SPS nicht direkt verarbeitet werden können. Praktisch wird also die Ausführung der Operation *value* zum Abschluss der Abfrage erzwungen, damit das Ergebnis in einen verarbeitbaren Wertebereich fällt.

In Abbildung 5.3 wird die Ausführung einer Abfrage durch einen Funktionsbaustein beispielhaft gezeigt. Der Bausteintyp query, der die Abfrage ausführt, ist willkürlich für das Beispiel gewählt und soll hier nur zur Illustration des Konzepts dienen; er wird später genau spezifiziert. Der Baustein hat einen Ausgang, der das Ergebnis einer Abfrage liefert, und einen Ausgang, der das Eintreffen eines neuen Ergebnisses für die Dauer eines Zyklus anzeigt. Die Abfrage qXisTypeZ, die zuvor durch die Funktionsbausteine in Abbildung 5.2 gebildet wurde, wird im Beispiel bei steigender Flanke am Eingang exc ausgeführt und das Attribut Id der resultierenden Tupel abgefragt. Wenn durch eine Abfrage neue Daten erhalten wurden und das Ergebnis nicht leer ist, wird die Variable XisTypeZ auf *true* gesetzt, sonst auf *false*.

Für die genaue Spezifikation der Funktionsbausteine entsprechend diesem Konzept werden in den nächsten Abschnitten die Bausteintypen beschrieben. Der wesentliche Punkt dabei ist die Unterscheidung zwischen Funktionsbausteinen zur Formulierung und zur Durchführung von Abfragen.

5.4.2. Funktionsbausteine für Abfrageoperationen

In den Abschnitten 5.2 und 5.3 wurden die Abfrageoperationen definiert, die nun durch Funktionsbausteine für den Anwender verfügbar gemacht werden sollen. Bei der Definition der Operationen wurden jeweils eine Definitionsmenge und Zielmenge angegeben. Dadurch lassen sich die Schnittstellen der Bausteine direkt herleiten. Die Mengen, mit denen die Definitions- und Zielmengen beschrieben wurden, also beispielsweise die

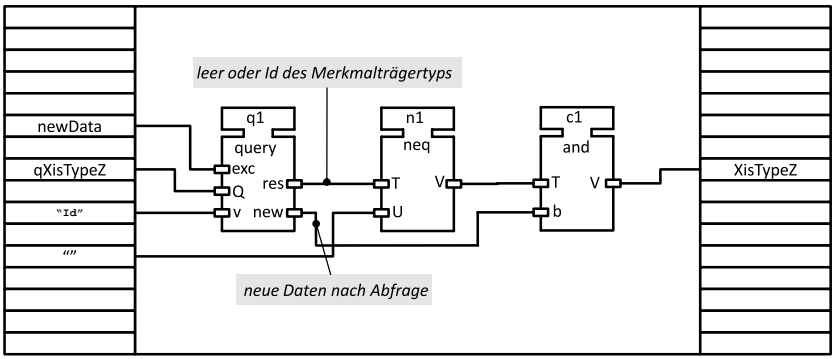

Abbildung 5.3.: Beispiel für die Ausführung einer Abfrage und Verarbeitung des Ereignisses durch Funktionsbausteine.

Menge aller Tupel oder die Menge aller Merkmalträger, sind natürlich keine bekannten Variablentypen innerhalb einer SPS. Das ist an dieser Stelle aber keine zwingende Voraussetzung, weil die Funktionsbausteine keine Daten innerhalb der SPS austauschen, sondern nur zum Aufbau von Abfragen dienen. Die Schnittstellen der Bausteine arbeiten daher immer mit dem Datentyp STRING, während davon ausgegangen wird, dass durch die Strings Daten vom korrekten Datentyp der Operation identifiziert werden. In Abbildung 5.2 liegt beispielsweise am Eingang b des Bausteins c1 vom Typ choose der String „Id=ProduktZ“ an, der eine boolesche Formel beinhaltet. In der Definition von *choose* wurde dementsprechend der Typ $\mathbb{BF}$ für die Menge der booleschen Formeln festgelegt. Das bedingt natürlich, dass sich der Wert des Strings, der am Eingang b anliegt, als korrekte boolesche Formel interpretieren lässt. Die Interpretation wird dann von demjenigen System durchgeführt, das die Abfrage als Dienstleister auswertet. Seitens der SPS ist die Verwendung des Datentyps STRING daher ausreichend, während der Dienstleister jeden Bestandteil einer Abfrage überprüfen muss. Innerhalb der SPS kann beim Aufbau einer Abfrage kein technischer Fehler auftreten, sondern es können nur ggf. nicht interpretierbare Abfragen aufgebaut werden.

Operativ verhalten sich die Bausteine so, dass bei jeder zyklischen Ausführung aus den Werten der Bausteineingänge ein einzelner String aufgebaut wird, der einen entsprechenden Aufruf der Operation textuell wiedergibt. Im Wesentlichen handelt es sich also um eine Konkatenation von Strings. Für einen Anwender ist dabei lediglich zu beachten, dass die Bausteine in der richtigen Reihenfolge ausgeführt werden, d.h. wenn ein Ausgang eines Bausteins b1 mit dem Eingang eines Bausteins b2 verbunden ist, dann muss b1 vor b2 ausgeführt werden. Anderenfalls kann der Aufbau einer syntaktisch und semantisch korrekten Abfrage nicht sichergestellt werden. Diese Bedingung ist aus Anwendersicht gut nachvollziehbar, wenn der Aufbau einer Abfrage wie eine Reihenschaltung von Filtern betrachtet wird, durch die schrittweise aus einer Informationsmenge die gesuchte Information herausgefiltert wird. Mit dieser Sichtweise wird auch klar, dass die

Verwendung von Rückwärtsschleifen nicht erlaubt ist. Die prototypische Implementierung (Kapitel 6) geht hier noch einen Schritt weiter: Bei der Ausführung von Abfragen werden die betreffenden Funktionsbausteine automatisch in der richtigen Reihenfolge ausgeführt und auf Rückwärtsschleifen überprüft. Dies ist jedoch nur aufgrund der im Prototyp gewählten Basistechnologie möglich und kann nicht für den allgemeinen Fall vorausgesetzt werden.

Die Art, in der der String für Abfragen aufgebaut wird, kann systemspezifisch angepasst werden. Wenn die Ressourcen des Laufzeitsystems stark begrenzt sind oder es mutmaßlich sehr viele und komplexe Abfragen im System geben wird, sind ggf. Vereinfachungen sinnvoll. Die Bezeichner von Operationen können statt ausgeschriebener Namen beispielsweise durch Kürzel ersetzt werden. Außerdem kann mit geschicktem Einsatz von Zeigern in der Implementierung das mehrfache Kopieren von Strings umgangen werden. Diese Optionen hängen aber stark vom jeweils eingesetzten Laufzeitsystem ab und lassen sich deshalb nicht allgemein festlegen.

Durch die Definition der Operationen, die Konvention zur Benutzung des Datentyps STRING und das einfache Verhalten ist die Spezifikation der Funktionsbausteintypen trivial. An dieser Stelle werden die Bausteintypen deshalb nur informell aufgelistet (s. Abbildung 5.4), wobei die Funktionsbausteine AGGREACH und AGGRALL die Operationen $aggrEach'$ und $aggrAll'$ implementieren. Die formelle Spezifikation der Implementierung befindet sich in Anhang A.

5.4.3. Funktionsbaustein zur Ausführung von Abfragen

Gemäß Anforderung 8 soll sich der Baustein zur Ausführung von Abfragen eng am Baustein READ der IEC 61131-5 orientieren. Schnittstellen und Funktion dieses Bausteins wurden in Abschnitt 3.4.3 vorgestellt. Der Bausteintyp wird nun so angepasst, dass er sich zur Ausführung von Abfragen eignet. Dieser neue Bausteintyp erhält die Bezeichnung QUERY.

Schnittstellen des Funktionsbausteins QUERY

Der READ-Baustein besitzt einen Eingang für das Signal zur Ausführung (REQ), einen Eingang zur Identifikation des Kommunikationskanals (ID) und ggf. mehrere Eingänge zur Identifikation der zu lesenden Daten (VAR_X). Im QUERY-Baustein wird in jedem Fall ebenfalls der Eingang REQ zur gesteuerten Ausführung von Abfragen benötigt. Außerdem muss er statt der abzufragenden Variablen VAR_X einen STRING-Eingang für die auszuführende Abfrage Q besitzen. Die Angabe eines Kommunikationskanals ist dagegen unnötig, wenn angenommen wird, dass immer derselbe Dienstleister zur Ausführung von Abfragen verwendet wird. Dieser kann dann durch eine globale Variable identifiziert werden und braucht nicht für jede Instanz des QUERY-Bausteins wiederholt zu werden.

Um das Ergebnis einer Abfrage weiterverarbeiten zu können, muss aus dem Tupel, das das Ergebnis der Abfrage ist, ein konkreter Wert ausgewählt werden. Dieser Wert muss in einem von der SPS verarbeitbaren Datentyp darstellbar sein. Typischerweise wird es sich bei dem gefragten Wert um den Wert einer Aussage über ein Merkmal handeln, aber allgemein muss die Möglichkeit bestehen, ein beliebiges Attribut zu identifizieren. Dazu erhält der QUERY-Baustein den Eingang VAR vom Typ STRING, durch den

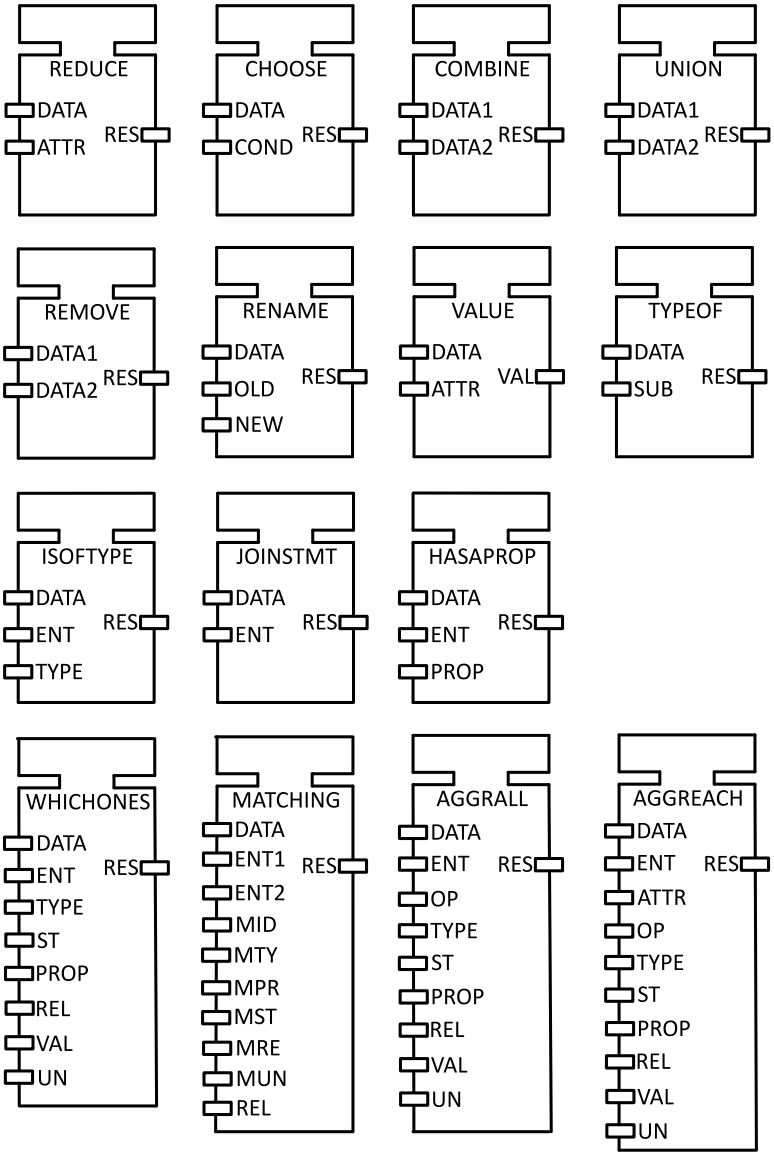

Abbildung 5.4.: Bibliothek der Funktionsbausteine für Abfrageoperationen.

das Attribut identifiziert wird. Die Benennung VAR (statt beispielsweise ATTR) stimmt zwar nicht mit der Terminologie des genutzten relationalen Modells überein, hier wird aber der Übereinstimmung mit dem READ-Baustein Vorrang gegeben.

Für den Fall, dass der Wert einer spezifischen Aussage über ein bestimmtes Merkmal eines Merkmalträgers oder eines Merkmalträgertyps erfragt werden soll, enthält der Baustein die Eingänge PROP, ST, REL und UN. Durch diese Eingänge können das betreffende Merkmal, die Aussageart, die Relation und die Einheit angegeben werden. Dadurch ermöglicht der QUERY-Baustein auch den direkten Zugriff auf Werte von Aussagen, ohne dass vorgelagerte Funktionsbausteine wie JOINSTMT verwendet werden müssen. Voraussetzung für diese Funktion ist natürlich, dass das Ergebnis der Abfrage am Eingang Q entweder Merkmalträger oder Merkmalträgertypen beinhaltet, damit diese nach den angegebenen Aussagen durchsucht werden können. Andere Tupel, die keine Merkmalträger oder Merkmalträgertypen bezeichnen, werden ignoriert. Die Verwendung des Eingangs PROP hat Priorität vor dem Eingang VAL: Wenn beide belegt sind, wird nach einer „passenden“ Aussage gesucht anstatt das durch VAL angegebene Attribut der Tupel auszugeben.

Die Ausgänge des Funktionsbausteins QUERY sind identisch mit einem READ-Baustein, der einen Datenausgang besitzt. Das bedeutet insbesondere, dass der RD-Ausgang eine beschreibbare Variable vom Typ ANY ist. Abbildung 5.5 zeigt den Funktionsbausteintyp. Die Spezifikation des Bausteins befindet sich in Anhang A.

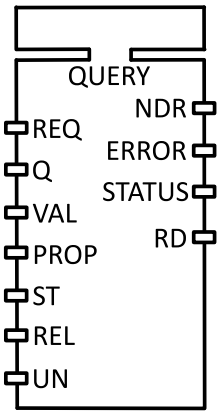

Abbildung 5.5.: Funktionsbausteintyp QUERY.

Verhalten des Funktionsbausteins QUERY

Für das Verhalten des Bausteins wird das Verhalten des Funktionsbausteintyps READ übernommen. Es entspricht dem in Abbildung 3.4 (Seite 47) gezeigten Ablauf mit an-

gepassten Variablennamen. Betrachtet man von außen her die Funktion von READ und QUERY, dann ist ein praktisch identisches Verhalten auch naheliegend: In beiden Fällen wird an einen Dienstleister eine Zeichenkette übergeben, die Daten identifiziert. Die Zeichenkette wird vom Dienstleister interpretiert und er schickt eine Antwort auf die Abfrage zurück, so dass die Antwort zu einem späteren Zeitpunkt am Funktionsbaustein zur Verfügung steht. Unterschiede betreffen nur den Inhalt von Abfrage und Antwort.

Wegen der weitgehenden Übereinstimmung und den nur unwesentlichen Unterschieden wird die Ausführungslogik des Funktionsbausteins an dieser Stelle nicht erneut erläutert. Eine Spezifikation befindet sich in Anhang B.

5.5. Systemarchitektur

5.5.1. Positionierung in der Automatisierungspyramide

Die vorgeschlagene Lösung besteht aus zwei technisch getrennten Komponenten: Den Funktionsbausteinen zur Formulierung und Ausführung von Abfragen und dem Dienstleister, der die Abfragen interpretiert und die gesuchte Information aus unterschiedlichen Datenquellen zusammenstellt. Für die Funktionsbausteine ist klar, dass sie innerhalb einer SPS, die typischerweise zu einem (Prozess-) Leitsystem gehört, ausgeführt werden. Innerhalb der Automatisierungspyramide befindet sich diese technische Komponente somit auf der Prozessleitebene.

Der Dienstleister, im Folgenden der Einfachheit halber als „Merkmaldienst“ bezeichnet, kann dagegen viel freier positioniert werden. Für den Merkmaldienst muss zugesichert sein, dass der QUERY-Funktionsbaustein Daten mit ihm austauschen kann und dass der Datenzugriff auf die diversen Quellen von Merkmalinformation möglich ist. Außerdem müssen natürlich die notwendigen Ressourcen an Speicher und Rechenzeit vorhanden sein. Es ergeben sich daraus drei mögliche Szenarien: Der Merkmaldienst kann direkt in der SPS bzw. im selben Hardwaresystem implementiert sein, er kann in einem nahen PC-System wie einer Operator Station implementiert sein, oder er kann als selbständiger Server in der Produktionsleitebene implementiert sein. Jede dieser Möglichkeiten kann in speziellen Anwendungsfällen die optimale Lösung sein, in den meisten Fällen ist jedoch die Implementierung als eigenständiger Server die beste Variante. Erstens wird so der normale Betrieb von Operator Station und SPS ganz sicher nicht durch die Verwendung zusätzlicher Ressourcen beeinflusst (s. Anforderung 10), während bei einem selbständigen Server die Ressourcen frei skalierbar sind. Zweitens muss der Merkmaldienst mit unterschiedlichen Datenquellen arbeiten und entsprechend unterschiedliche Datenschnittstellen implementieren. Die Datenquellen befinden sich überwiegend in der Produktionsleitebene. Ein Merkmaldienst als Server in der Produktionsleitebene kann also direkt mit diesen Datenquellen kommunizieren und muss nicht zwischen zwei Ebenen überbrücken, die ggf. unterschiedliche Teilnetze betreiben. Die Kommunikation mit den Funktionsbausteinen ist dagegen inhaltlich einfach – es müssen nur Strings und einfache Werte übertragen werden, was auch zwischen Prozessleit- und Produktionsleitebene mit etablierten Protokollen wie OPC DA/UA oder ACPLT/KS problemlos funktioniert. Aus diesen Gründen wird hier davon ausgegangen, dass der Merkmaldienst als selbständiger Server implementiert wird.

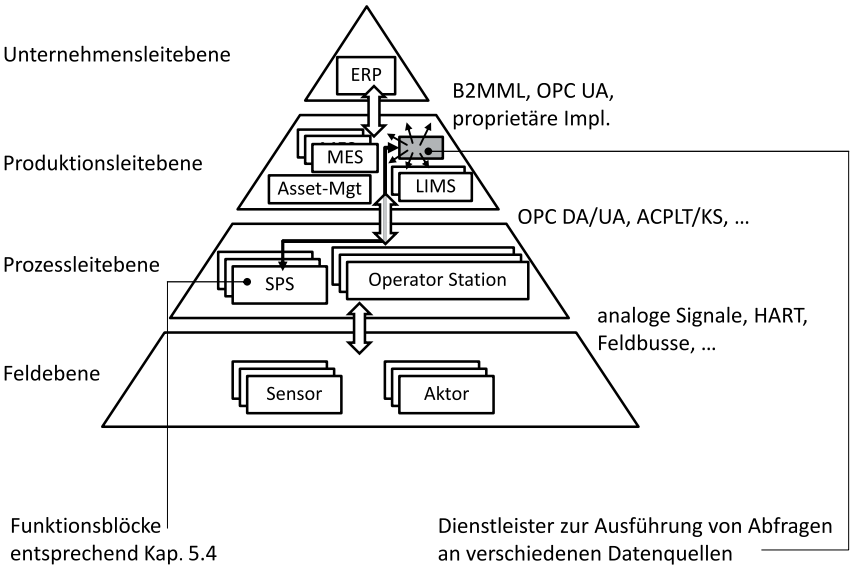

Abbildung 5.6.: Einordnung des Dienstes für Merkmalabfragen in der Automatisierungspyramide.

5.5.2. Komponenten des Dienstes für Merkmalabfragen

In Abbildung 5.6 ist der Merkmaldienst nur als graues Kästchen dargestellt, das mit anderen Systemen kommuniziert. In diesem Abschnitt wird der Merkmaldienst nun genauer spezifiziert. Das geschieht auf dem Abstraktionsniveau von Softwarekomponenten; genauere Details auf dem Niveau von Klassen werden der individuellen Implementierung überlassen. Einen Überblick in Form eines UML-Komponentendiagramms zeigt Abbildung 5.7.

Entsprechend dem Konzept eines Dienstes ist es für den Dienstnutzer grundsätzlich irrelevant, wie der Dienst intern implementiert ist. Für eine detaillierte Spezifikation müssten auch viele Details des Anwendungsszenarios berücksichtigt werden, beispielsweise

- die genutzten Kommunikationsnetze,
- die genutzten Kommunikationsprotokolle,
- die Technologie der Datenquellen,
- das Datenmodell der Datenquellen,
- die für den Merkmaldienst verfügbare Hard- und Software,
- die für die Implementierung genutzte Technologie.

Die dadurch entstehende kombinatorische Vielfalt wird hier dadurch beantwortet, dass einige Komponenten je nach Anwendungsfall auswechselbar sind und dass für fest vorhandene Komponenten nur die Funktionalität beschrieben wird, nicht durch welche Datenstrukturen und Algorithmen diese Funktionalität geschaffen wird. Beim Merkmaldienst kann folglich zwischen drei Hauptbestandteilen unterschieden werden. Die Kernkomponenten bilden das Rückgrat der Anwendung und sind unverändert in jeder Installation vorhanden. Plug-Ins sind Komponenten, die je nach Anwendungsfall in einer einzelnen Installation vorhanden sein können. Für unterschiedliche Datenquellen werden unterschiedliche Plug-Ins verwendet. Diese können die externen Quellen entweder dynamisch anbinden oder deren Inhalt einmalig in ein internes Datenmodell laden. Den dritten Hauptbestandteil bilden Konfigurationsdateien, in denen Installationsspezifika konfiguriert werden – beispielsweise Adressen von Datenservern oder Dateinamen. Die Konfigurationsdateien können zur Laufzeit des Systems verändert werden.

Kernkomponenten

Zentrale und wichtigste Komponente des Merkmaldienstes ist die Komponente „CoreServer“ (der Name, wie auch die übrigen Komponentennamen, wurde als illustratives Beispiel entsprechend üblichen Benamungskonventionen der Softwareentwicklung gewählt). CoreServer stellt ein intern verwendetes Datenmodell bereit und implementiert die Operationen der Abfragesprache entsprechend Kapitel 5.2 und 5.3 auf diesem Datenmodell.

Datenabfragen erreichen CoreServer als String, der zunächst geparst werden muss, um die auszuführenden Operationen zu dekodieren. Bei der Ausführung der Operationen müssen dann zwei Fälle unterschieden werden: Die Daten, auf denen die Abfrage ausgeführt wird, liegen entweder im internen Datenmodell vor oder in einem

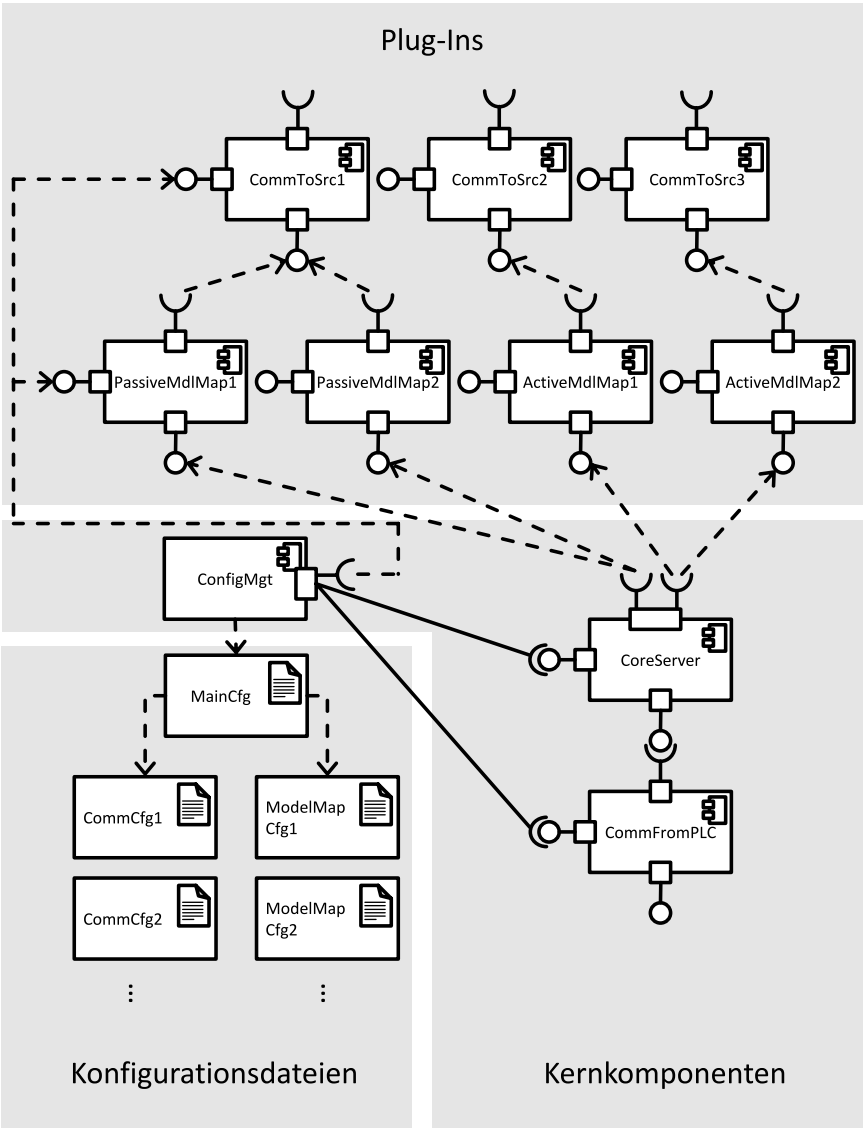

Abbildung 5.7.: Komponenten des Dienstes für Merkmalabfragen als UML-Komponentendiagramm.

externen Datenspeicher wie einer Datenbank. Dementsprechend kann CoreServer die Abfrage selbst ausführen oder delegiert sie an eine Plug-In -Komponente, die den externen Datenspeicher anbindet. Die Implementierung der Abfrageoperationen in CoreServer kann also durch spezialisierte Implementierungen der einzelnen Plug-Ins ersetzt werden. Für die Implementierung der Abfragen ist die parallelisierte Datenverarbeitung durch Threads möglich und kann zur Beschleunigung der Verarbeitung beitragen. Entsprechend der Grundidee eines Dienstes ist das erlaubt, sofern die Abfragen eines Dienstnutzers nicht in falscher Reihenfolge bearbeitet werden (s. Abschnitt 2.1.2, S. 15). Wenn also beispielsweise ein Nutzer A Daten aus einer langsamen Datenquelle abfragt und unmittelbar danach ein anderer Benutzer B Daten aus einer schnellen Datenquelle abfragt, dann würde B nicht von A blockiert. Aus technischer Sicht wäre es natürlich auch möglich, dass Abfragen desselben Benutzers im Sinne einer Bearbeitung „so schnell wie möglich“ ohne Rücksicht auf die Reihenfolge bearbeitet werden könnten. Das Problem daran wäre aber, dass der Benutzer dann in einigen Fällen die Abfragen innerhalb der SPS synchronisieren müsste. Beispielsweise könnte in einem SPS-Programm erst die aktuelle Auftragsnummer und dann ein Merkmal dieses Auftrags abgefragt werden. Wenn der SPS-Programmierer sich nun nicht darauf verlassen kann, dass die aktuelle Auftragsnummer vor der Abfrage des Merkmals bekannt ist, müsste ein entsprechender Mechanismus zur Absicherung implementiert werden. Solche Implementierungen sind in einer SPS umständlich und damit auch fehleranfällig. Eine Abweichung vom Grundprinzip der reihenfolgegemäßen Abarbeitung pro Benutzer ist daher nicht gestattet. Sofern die Voraussetzung dafür, dass Benutzer (d.h. einzelne SPS) eindeutig identifiziert werden können, nicht erfüllt ist, müssen alle Abfragen in Reihenfolge des Eintreffens bearbeitet werden.

Das Datenmodell in CoreServer ist eine Abbildung der Relationenschemas aus Abschnitt 5.1 und entspricht damit auch dem logischen Modell aus Abschnitt 2.1.1. Dies kann beispielsweise in objektorientieren Sprachen so realisiert werden, dass Relationenschemas als Klassen implementiert werden und Tupel durch deren Instanzen abgebildet werden, für die es einen zentralen Index gibt. Für große Datenmengen bietet sich hier auch die sogenannte objektrelationale Abbildung an, eine Technologie, mit der objektorientierte Datenmodelle physisch in Datenbanken gespeichert werden (z.B. Hibernate für Java oder NHibernate für .NET).

Die Kommunikation mit der SPS wird durch die Komponente „CommFromPLC“ realisiert. Sie kann beispielsweise durch einen OPC UA oder ACPLT/KS Server realisiert werden, der die Abfrage-Strings annimmt und Antworten zurück an die SPS sendet. Die spezielle Kommunikationstechnologie ist im einzelnen Einsatzfall mit großer Wahrscheinlichkeit fest vorgegeben, so dass CoreServer und CommFromPLC über eine Schnittstelle fest miteinander verbunden sind. Bei einer konkreten Implementierung kann auch davon ausgegangen werden, dass vorhandene Bibliotheken der jeweiligen Kommunikationstechnologie genutzt werden können, so dass die wesentliche Funktionalität von CommFromPLC nur darin besteht, diese Kommunikationsbibliotheken für den hier auftretenden Anwendungskontext zu kapseln.

Dritte Kernkomponente ist ConfigMgt. Die Aufgabe von CfgMgt ist, sämtliche Konfigurationsinformation des Merkmaldienstes zu verwalten und bei Bedarf zu verändern. Dazu kann der Anwender die Konfigurationsdaten in Textdateien ablegen. Es gibt eine Hauptdatei, hier „MainCfg“ genannt, die immer notwendige Informationen beinhaltet. Beispielsweise sind die Konfigurationsinformationen für CoreServer und CommFrom-

PLC in der Datei MainCfg abgelegt. In der Datei sind außerdem Verweise auf weitere Dateien gespeichert, in denen Daten für die Plug-In-Komponenten gespeichert sind. Diese Verteilung von Information auf unterschiedliche Dateien unterstützt die Flexibilität der Anwendung und damit eine flexible Datenanbindung der SPS entsprechend Anforderung 11. Auf diese Art sind Konfigurationsdaten für die einzelnen Datenquellen unabhängig voneinander und können ohne Wechselwirkung verändert werden. Außerdem kann jedes Plug-In sein jeweils sinnvollstes Datenformat verwenden. Nachdem Konfigurationsdaten verändert wurden, teilt der Anwender ConfigMgt mit, dass die Konfiguration aktualisiert werden muss. Die veränderten Dateien können dann leicht erkannt werden und es werden nur die betroffenen Komponenten aktualisiert. Die Schnittstelle zum Anwender kann beispielsweise eine einfache Kommandozeilenschnittstelle sein, wie auch bei der Administration von Web-Servern üblich. Die Komponenten innerhalb der Merkmaldienstes besitzen jeweils ein Interface für das Einspeisen von Konfigurationsinformation (in Abbildung 5.7 jeweils links eingezeichnet).

Plug-In -Komponenten

Plug-In -Komponenten können anwendungsfallspezifisch in das System integriert werden. Aktuelle Softwaretechnologien unterstützen auch das Nachladen von Bibliotheken, so dass Plug-Ins auch noch zur Laufzeit hinzugefügt (oder entfernt) werden können (s. Anforderung 11). Die Plug-Ins werden in zwei Arten unterteilt; im Komponentendiagramm sind sie daher in zwei Zeilen dargestellt.

Die mit „CommToSrcX“ bezeichneten Plug-Ins (oben dargestellt) realisieren die reine Datenverbindung zu den verwendeten Datenquellen. Beispielsweise kann es ein solches Plug-In für XML-dateibasierte Quellen geben, eines für SQL-Datenbanken usw. Die Plug-Ins können dabei aufeinander aufbauen, so dass ein Plug-In eine speziellere Form des Datenzugriffs realisiert als ein anderes. In diesem Fall sind es zwar getrennte Komponenten, die jedoch eine Abhängigkeit besitzen.

Die zweite Art von Plug-Ins bildet das Datenmodell der Datenquelle auf die intern benutzten Abfrageoperationen und das interne Datenmodell ab. Dabei kann zwischen einer aktiven und einer passiven Abbildung unterschieden werden. Bei der passiven Abbildung wird die Datenquelle einmalig von der Komponente eingelesen und im Datenmodell von CoreServer abgelegt. Abfragen werden dann von CoreServer selbst auf den Daten durchgeführt. Über ConfigMgt kann der passiven Plug-In-Komponente mitgeteilt werden, dass bei einer Aktualisierung die Daten neu eingelesen werden müssen. Diese Art von Plug-In eignet sich für kleine, dateibasierte Datenquellen.

Aktive Plug-Ins führen Abfragen dagegen selbst aus bzw. übersetzen Abfragen so, dass die Datenquelle sie ausführt. Entsprechend eignet sich diese Art Plug-In zur Anbindung von Datenbanken. Wie oben beschrieben, muss CoreServer für jede (Teil-) Abfrage entscheiden, ob die Abfrage auf dem internen Datenmodell ausgeführt wird oder an ein aktives Plug-In delegiert werden kann. Dies kann danach entschieden werden, ob die Daten von einem aktiven oder passiven Plug-In zur Verfügung gestellt werden. Entsprechend besitzen aktive und passive Plug-Ins auch unterschiedliche Schnittstellen.

Die „CommToSrcX“ und „Active/PassiveMdlMapX“ -Plug-Ins können je nach Bedarf miteinander kombiniert werden, weil inhaltlich unterschiedlich aufgebaute Modelle in technisch gleichen Datenquellen abgelegt sein können. Für die Anbindung von SQL-Datenbanken wird beispielsweise nur ein Plug-In benötigt, auf das dann mehrere aktive

Plug-Ins zur Modellabbildung aufbauen können. Diese Abhängigkeiten müssen dem Anwender bewusst sein und er muss die jeweils notwendigen Plug-Ins laden.

6. Prototypische Implementierung

Zur Validierung des in Kapitel 5 beschriebenen Konzepts wurde eine prototypische Implementierung entwickelt. In diesem Kapitel werden die wichtigsten technischen Aspekte dieser Implementierung erläutert. Der folgende Abschnitt befasst sich mit den genutzten technischen Grundlagen. Ziel ist dabei nur, ein ausreichendes Verständnis der Technologien zu vermitteln, so dass die Funktionsweise des Prototypen und dessen Umfang nachvollzogen werden können. Für tiefergehende Details wird auf die jeweils angezeigte Literatur verwiesen.

Der Abschnitt 6.2 beschreibt wichtige Eigenschaften der Architektur der Implementierung. Auch hier werden nicht alle Details erläutert, sondern nur solche Aspekte, die für das Funktionieren des Gesamtsystems in Hinblick auf die Anforderungen relevant sind (siehe Kapitel 4). Abschließend wird die Verwendung des Prototypen in Anwendungsbeispielen illustriert.

6.1. Technische Grundlagen

6.1.1. Die Laufzeitumgebung ACPLT/OV

ACPLT/OV ist eine Laufzeitumgebung für Automatisierungssysteme [55]. Die Bezeichnung „ACPLT/OV“ setzt sich zusammen aus einer Abkürzung für den Lehrstuhl für Prozessleittechnik der RWTH Aachen (Aachener Prozessleittechnik) und der Abkürzung für „Objektverwaltung“. Entsprechend wurde das System an dem genannten Lehrstuhl entwickelt und die Verfügbarkeit von Objektorientierung in der Laufzeitumgebung ist eine der herausragenden technischen Eigenschaften.

Basis von ACPLT/OV ist die Programmiersprache C bzw. das normierte ANSI C [4]. Dadurch kann das System auf vielen einfachen Hardwaresystemen verwendet werden, für die nur C-Compiler existieren. Durch die Basisbibliotheken von ACPLT/OV wird das Programmieren entsprechend dem Paradigma der Objektorientierung ermöglicht. Zum Konzept von ACPLT/OV gehört daher ein Metamodell, das die Struktur der objektorientierten Modelle vorgibt. Beispielsweise sieht das Metamodell eine baumförmige Organisation der Objekte zur Laufzeit vor, in der jedes Objekt eine lokal (d.h. unter seinen Geschwistern) eindeutigen Namen hat. Das Konzept der Vererbung ist ebenfalls verfügbar.

Praktisch geschieht die Definition von Klassen und Beziehungen zwischen Instanzen der Klassen durch eigene Beschreibungsdateien, weil C hierfür keine nativen Sprachmittel hat. Durch spezielle Tools der Laufzeitumgebung wird dann aus den Beschreibungsdateien C-Code generiert, innerhalb dessen die spezifizierten Methoden der Klassen implementiert werden. Durch spezielle Datenstrukturen sind dabei auch Informationen über die Klassen und Instanzen zur Laufzeit verfügbar, so dass die Objektmodelle zur Laufzeit erkannt und verändert werden können.

ACPLT/OV bietet einige spezifische Eigenschaften aufgrund der Zielsetzung als Basissystem für die Automatisierung. Dazu gehört auch, dass die Ausführung von Programmen analog zur Arbeitsweise einer SPS möglich ist. Das Prinzip der zyklischen Ausführung von Funktionen, beispielsweise von Funktionsbausteinen, ist daher im System fest verankert. Dies wird von den im Folgenden beschriebenen Bibliotheken ACPLT/FB und ACPLT/KS genutzt. Durch ACPLT/OV kann jedoch keine vollständige SPS implementiert werden – beispielsweise wird das Prinzip der globalen oder lokalen Sichtbarkeit von Variablen nicht unterstützt (vgl. Abschnitt 3.4.1) und Echtzeitfähigkeit ist nicht nativ vorhanden. Der Unterschied in der Programmierung für eine SPS und der Programmierung für ACPLT/OV ist für die hier beschriebene Implementierung des Prototyps größtenteils nicht relevant. Dort, wo es Unterschiede gibt, wird darauf entsprechend hingewiesen werden.

6.1.2. Die Bibliothek ACPLT/FB

ACPLT/FB steht für das ACPLT „Funktionsbausteinsystem“, eine Bibliothek für ACPLT/OV zur Implementierung von Funktionsbausteinen [28, 69]. Durch die verfügbare Objektorientierung können Klassen dritter Bibliotheken von der Klasse „functionblock“ aus ACPLT/FB erben. Dadurch sind viele der Konzepte von Funktionsbausteinen im Sinne der IEC 61131-3 [21] in der erbenden Klasse verfügbar, so dass auf einfache Art neue Funktionsbausteintypen implementiert werden können. Beispielsweise können Variablen der neuen Klassen durch spezielle Flags als Ein- oder Ausgänge von Funktionsbausteinen deklariert werden. ACPLT/FB stellt dann die für das Engineering von Signallinien zwischen Funktionsbausteinen notwendigen Funktionen bereit und sorgt während der Ausführung für den Signaltransport.

Durch die Möglichkeit von ACPLT/OV, Funktionen zyklisch aufzurufen, bietet ACPLT/FB sogenannte Tasklisten für die zyklische Ausführung von Funktionsbausteinen an. Jeder Funktionsbaustein kann, muss aber nicht in eine Taskliste eingehängt werden. Im zweiten Fall wird der Funktionsbaustein entweder durch einen anderen Mechanismus oder überhaupt nicht ausgeführt. Jede Taskliste enthält eine Menge von Funktionsbausteinen in fester Reihenfolge, die in dieser Reihenfolge zyklisch ausgeführt werden. Die Taktung der Ausführung wird pro Taskliste festgelegt. Tasklisten entsprechen dem in der IEC 61131-3 beschriebenen Konzept „Task“ (vgl. Abschnitt 3.4.1). Die Ausführungszeit jedes Funktionsbausteins wird während jeder Ausführung gemessen und mit einer oberen Schranke verglichen. Weil Echtzeitfähigkeit nicht zwangsläufig vorhanden ist – das System wird nicht zwangsläufig mit einem Echtzeitbetriebssystem ausgeführt – werden Funktionsbausteine während der Ausführung nicht unterbrochen. Stattdessen wird bei Überschreitung einer Zeitschranke eine Warnung ausgegeben.

Die Implementierung der Ausführungslogik eines Funktionsbausteintyps geschieht in genau einer Funktion pro Funktionsbausteintyp, genannt „typemethod“. Innerhalb der typemethod werden üblicherweise die Eingänge eines einzelnen Funktionsbausteins gelesen, Berechnungen ausgeführt und dann die Ausgänge des Funktionsbausteins beschrieben. Die typemethod wird durch die jeweilige Taskliste des Funktionsbausteins aufgerufen, kann aber auch explizit z.B. von anderen Funktionsbausteinen aufgerufen werden.

6.1.3. Das Kommunikationsprotokoll ACPLT/KS

ACPLT/KS bezeichnet einerseits ein Kommunikationsprotokoll auf Ebene 5 und 6 des ISO/OSI-Referenzmodells [46], ist aber gleichzeitig auch der Name verschiedener Implementierungen des Protokolls. KS steht für „Kommunikationssystem“. Technische Basis von ACPLT/KS sind die Standards TCP/IP [12, 13] und ONC/RPC [64]. Anwendungsgebiet von ACPLT/KS ist die industrielle Automatisierung, beispielsweise Prozessleitsysteme und Softwaresysteme im Umfeld der industriellen Produktion (siehe Abschnitt 3.3).

Das Protokoll setzt strukturierte Informationen in Form von Objekten und deren Eigenschaften (Variablen und Objektbeziehungen) voraus, wie sie beispielsweise in einem ACPLT/OV-System gefunden werden. Für den Zugriff auf die Information werden Basisoperationen bereitgestellt, durch die Objekte und deren Eigenschaften erstellt und verändert werden können. Die in einem Server verfügbaren Objekte können vollständig durch das Protokoll abgefragt werden, so dass die gesamte Information im Server ohne Vorwissen für einen Klienten erkennbar ist. Implementierungen von ACPLT/KS gibt es für das ACPLT/OV Laufzeitsystem, für zahlreiche Prozessleitsysteme, Datenarchive und als Bibliotheken in C++ und Java.

6.1.4. Das ACPLT-Dienstsystem

Das Dienstsystem ermöglicht die Nutzung von Diensten (s. Abschnitt 2.1.2) mit den ACPLT-Technologien. Es besteht aus Bibliotheken zur Bereitstellung und Nutzung von Diensten. Im Kontext der prototypischen Implementierung sind die technische Basis und die Ausführungslogik des Dienstsystems von Interesse und werden hier kurz beschrieben.

Das Dienstsystem funktioniert auf Basis von nachrichtenbasierter Kommunikation zwischen Dienstanbieter und -nutzer. Intern wird dazu ein Nachrichtenformat verwendet, das einen Header-Bereich für administrative Informationen (sendender Server, sendende Komponente im Server, empfangender Server, ...) und einen Body-Bereich für den Nachrichteninhalt vorsieht [30]. Die Nachrichten können beispielsweise über ACPLT/KS verschickt werden, es ist aber auch möglich die Nachrichten direkt über TCP, ohne erneute Kapselung in ein Protokoll der Anwendungsschicht, zu verschicken. Das Dienstsystem nutzt ein spezielles Nachrichtenformat, in dem zusätzlich zu den vorhandenen Header-Informationen auch der gerufene Dienst, die gerufene Operation und Parameter der Operation angegeben werden. Außerdem enthält jede Nachricht eine für den Absender eindeutige Id, so dass bei der Beantwortung eines Dienstaufrufs auf diese Id Bezug genommen werden kann.

Der Ablauf eines Dienstaufrufs beginnt mit dem Erstellen einer entsprechenden Nachricht beim Dienstnutzer. Die Nachricht wird an den anbietenden Server verschickt. Dort können Nachrichten entweder durch eine untere Protokollschicht wie ACPLT/KS direkt an die anbietende Komponente innerhalb des Servers weitergeleitet werden, oder es gibt eine zentrale Komponente, bei der alle Dienstanbieter im Server registriert sind. Im zweiten Fall „entpackt“ die zentrale Verwaltungskomponente die Aufrufparameter und ruft direkt die angefragte Operation auf. Innerhalb des Dienstsystems sind beide Varianten implementiert. Der Aufruf der Operation kann nun als synchroner Aufruf erfolgen, in dem die Operation vollständig ausgeführt wird und direkt nach der Fertig-

stellung eine Antwortnachricht zurückgeschickt wird (sofern eine Antwort erforderlich ist). Alternativ dazu kann die Operation die Ausführungskontrolle zurückgeben und die verantwortliche Komponente kann zu einem späteren Zeitpunkt über das Zurücksenden einer Antwortnachricht entscheiden. Das ist insbesondere bei Operationen mit langer Ausführungsdauer oder der Durchführung von Aktionen in der physischen Welt sinnvoll. Die Antwortnachricht hat wie die Aufrufnachricht ein spezielles Format und enthält eine Referenz auf die ursprüngliche Aufrufnachricht.

Innerhalb von ACPLT/OV gibt es eine spezielle Bibliothek, die den Dienstnutzer bei der Durchführung von Aufrufen unterstützt. Die Bibliothek baut Nachrichten für Dienstaufrufe auf und versendet sie, außerdem wird das Eintreffen von Antwortnachrichten überwacht und diese werden der aufrufenden Komponente zugeteilt.

6.2. Softwarearchitektur

6.2.1. Architektur des Klienten

In Abbildung 6.1 wird die grundlegende Architektur des Klienten in Form eines UML-Klassendiagramms gezeigt. Die erstellte Bibliothek für Funktionsbausteine trägt den Namen „PropertyInfos“. Die Bibliothek baut auf der in Abschnitt 6.1 beschriebenen Bibliothek ACPLT/FB auf und verwendet die ebenfalls in 6.1 beschriebene Bibliothek zur Durchführung von Dienstaufrufen. Die unten im Diagramm gezeigten Klassen ServiceClient_API und CallReply stammen aus dieser Bibliothek, die oben dargestellt Klasse functionblock aus ACPLT/FB.

Die Funktionalität der Funktionsbausteine ist mit Ausnahme des Bausteins QUERY gleich und wird nur durch die unterschiedlichen Eingänge spezifisch. Diejenigen Bausteine, aus denen Operationen zusammengesetzt werden, sind stellvertretend auf der rechten Seite in Abbildung 6.1 durch die Klassen REDUCE, CHOOSE und COMBINE dargestellt. Die Klassen der weiteren Funktionsbausteintypen aus Abbildung 5.4 (Seite 75) wurden aus Platzgünden ausgelassen. Im Diagramm werden Ein- und Ausgänge von Funktionsbausteinen durch nach innen bzw. außen gerichtete Pfeile dargestellt (als sinnvolle Ergänzung der üblichen UML-Notation für Klassendiagramme). Die Ein- und Ausgänge in der Implementierung entsprechen Abbildung 6.1. Die Ausführungslogik dieser Bausteine liest die Eingänge des Bausteins und schreibt die auszuführende Operation auf den Ausgang RES. Wenn beispielsweise an den Eingängen eines COMBINE-Bausteins die Werte „d1“ und „d2“ anliegen, würde das Ergebnis „COMBINE(d1,d2)“ auf den Ausgang RES geschrieben. Die Ausführungslogik ist jeweils in der typemethod implementiert. Alle rechts abgebildeten Funktionsbausteine erben von der abstrakten Klasse opBase. Dadurch können Instanzen dieser Funktionsbausteine im ACPLT/OV-Systems zur Laufzeit als ein Bausteintyp erkannt werden. Außerdem kann der Ausgang RES vererbt werden. Als Hinweis auf die spezielle Funktion von opBase ist der Name der Klasse nicht in Großbuchstaben geschrieben.

Der Funktionsbaustein QUERY bildet den Kern der Bibliothek. Er baut Abfragen auf, sendet sie an den ausführenden Dienst und verarbeitet die Antworten. Die notwendigen Konfigurationsinformationen, beispielsweise die IP-Adresse des Dienstes, werden von einer Instanz der Klasse config gelesen. Diese Instanz wird beim Laden der Bibliothek einmalig angelegt und mit Daten aus der Konfigurationsdatei des Servers beschrieben,

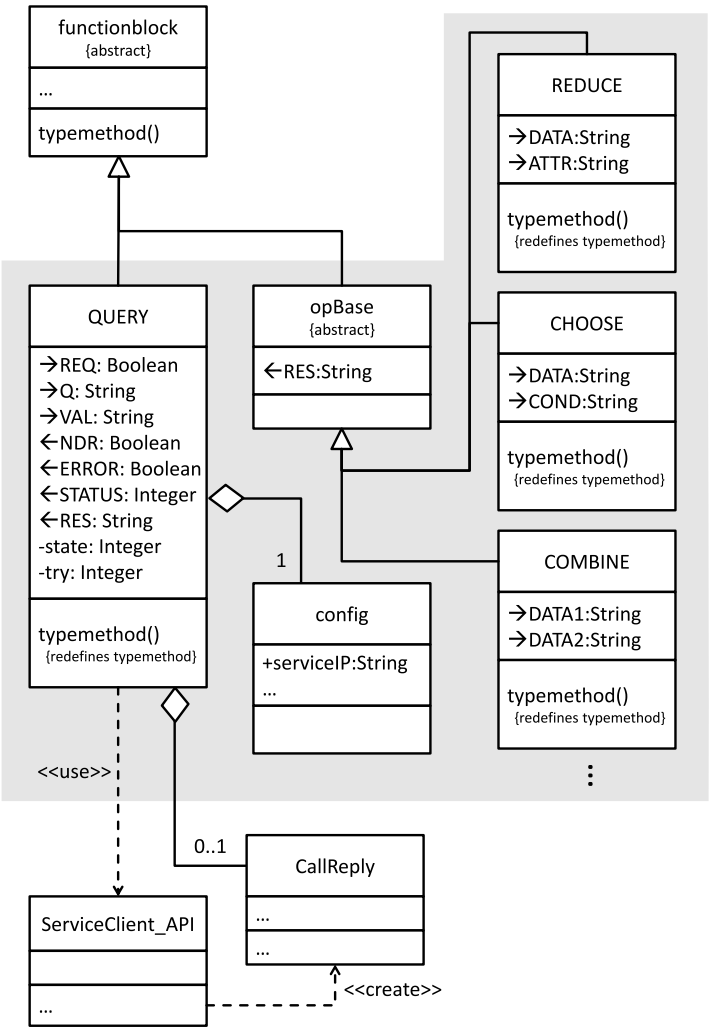

Abbildung 6.1.: Grundlegende Architektur der Klientenbibliothek (grau hinterlegt), dargestellt in der Notation eines UML-Klassendiagramms mit Pfeilen für Ein-/Ausgangssignale von Funktionsbausteinen.

sie kann aber auch später zur Laufzeit des Systems noch verändert werden. Jede Instanz von QUERY kennt dieselbe Instanz von config und nutzt die Konfigurationsdaten bei der Ausführung von Abfragen.

QUERY verwendet die Programmierschnittstelle ServiceClient_API zur Ausführung Dienstaufrufen. ServiceClient_API bietet Funktionen zur Erstellung und Ausführung von Dienstaufrufen, außerdem wird der Ausführungsstatus eines Aufrufs überwacht und ggf. ein Objekt vom Typ CallReply erzeugt, das der QUERY-Instanz zur Verfügung gestellt wird. Der genaue Ablauf wird in Abschnitt 6.2.3 erklärt, eine Spezifikation des Bausteins ist in Anhang A und B.

6.2.2. Architektur des Merkmaldienstes

Die Einteilung der Systemkomponenten des Merkmaldienstes wurde bereits in Abschnitt 5.5.2 vorgenommen (siehe auch Abbildung 5.7, Seite 80). An dieser Stelle soll nun die Architektur auf Klassenebene erläutert werden. Es wird aber nicht auf alle Klassen in allen Details eingegangen; stattdessen beschränken sich die Ausführungen auf die Anbindung von Plug-Ins und die Verwendung des internen Datenmodells. Die Anbindung der SPS und die Verwaltung von Konfigurationsdaten werden hier bewusst ausgelassen, weil sie für das Gesamtkonzept keine große Bedeutung haben und Details zur Implementierung keine neuen Einsichten brächten. Das gesamte Design ist stark von Anforderung 12 aus Kapitel 4 (Seite 53) geprägt, die nach einer einfachen Erweiterbarkeit für zusätzliche Datenquellen verlangt.

Ein Ausschnitt der Architektur des Merkmaldienstes wird in Abbildung 6.2 gezeigt. Entsprechend der Aufteilung der Komponenten aus Abschnitt 5.5.2 gibt es vier Pakete. In der Implementierung des Prototyps werden die Pakete durch Java-Packages realisiert. Das in Kapitel 2.1 eingeführte und in Kapitel 5 verwendete Metamodell für Merkmale ist im Paket model durch Java-Klassen modelliert. Diese Klassen dienen der Speicherung von Merkmaldaten, die aus statischen Informationsquellen, beispielsweise Dateien, geladen wurden. Diese Klassen sind nicht nach außen sichtbar. Jede Instanz einer dieser Klassen ist einer Instanz von DataSource zugeordnet, die im System die jeweilige Datenquelle repräsentiert.

Für unterschiedliche Datenquellen lassen sich passende Plug-Ins implementieren. Beispielhaft sind in Abbildung 6.2 die Plug-Ins XmlFileConnector und BmeCatMapper dargestellt, die sich im Paket plugins befinden. XmlFileConnector ist für den technischen Datenzugriff auf XML-Dateien verantwortlich, während BmeCatMapper das Datenmodell des BMEcat-Formats auf das interne Datenmodell aus dem Paket model abbildet. BMEcat ist ein XML-basiertes Datenformat für Katalogdaten, das beispielsweise für den Austausch von eCl@ss-konformen Produktbeschreibungen verwendet werden kann. BmeCatMapper kann mit Hilfe von XmlFileConnector diese Dateien einlesen und erstellt ein entsprechendes DataSource-Objekt mit den enthaltenen Modellen. Analog dazu können auch andere dateibasierte Datenquellen verwendet werden. Entscheidend ist, dass das jeweilige Plug-In zur Datenquelle die Merkmaldaten korrekt auf das interne Datenmodell abbildet. Der Aufwand für diese Abbildung ist überschaubar groß – im Fall von BmeCatMapper handelt es sich um ca. 200 Zeilen Java-Code.

Für die einheitliche Implementierung von Plug-Ins enthält das Paket plugin einige Interfaces, deren Implementierung für Plugins obligatorisch ist. Beispielsweise imple-

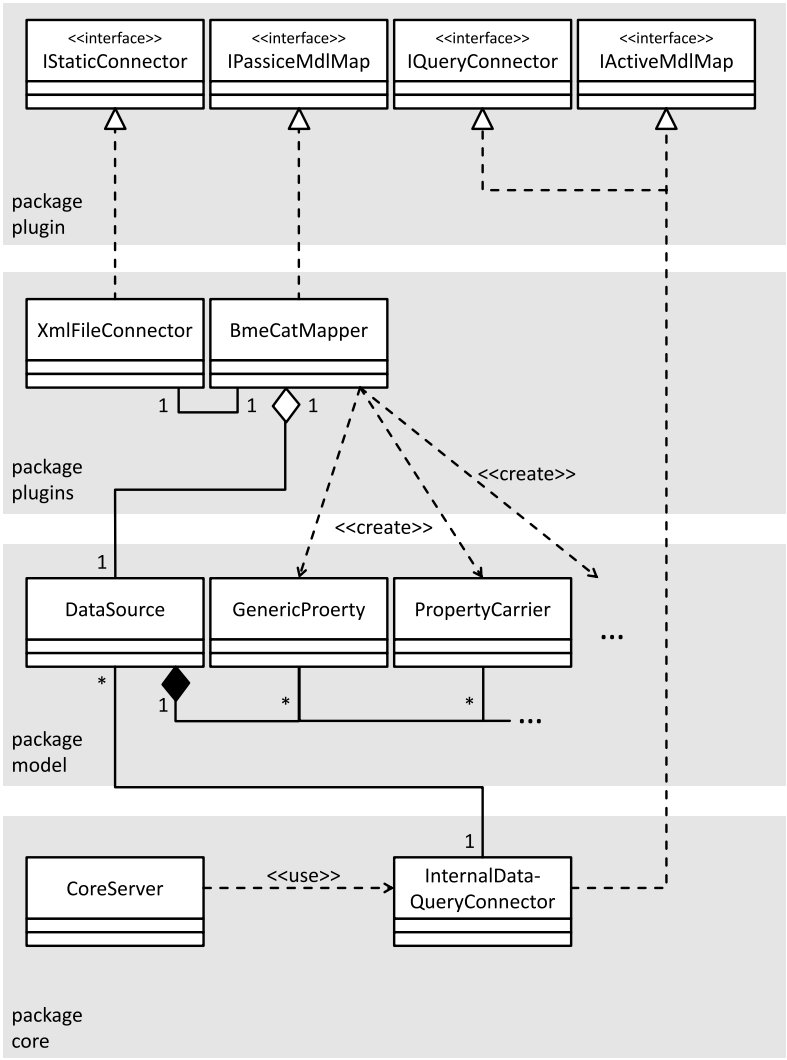

Abbildung 6.2.: Ausschnitt der Architektur des Merkmaldienstes mit Fokus auf Plug-Ins und das Datenmodell. Darstellung als vereinfachtes UML-Klassendiagramm.

mentiert XmlFileConnector das Interface IStaticConnector und BmeCatMapper implementiert IPassiveMdlMap. IStaticConnector definiert Methoden zur Übergabe von Konfigurationsdaten und zum Laden von Daten. IPassiveMdlMap definiert ebenfalls eine Methode zur Übergabe von Konfigurationsdaten (die aus den Konfigurationsdateien des Systems stammen), außerdem enthält IPassiveMdlMap Methoden für die Datenabbildung in das interne Datenmodell und für das Neuladen von Daten. Durch die Verwendung dieser Interfaces kann das System beim Start und im Betrieb uniform auf alle verwendeten Plug-Ins zugreifen und ihnen ihre spezifischen Konfigurationsinformationen übergeben.

Das Paket plugin enthält zwei weitere Interfaces: IQueryConnector und IActiveMdlMap. Diese Interfaces sind zur Anbindung von Datenquellen gedacht, die dynamisch sind und die daher jede Abfrage selbst ausführen, anstatt sie auf dem internen Datenmodell ausführen zu lassen. Solche Datenquellen können Datenbanken oder OPC UA -Server sein. IActiveMdlMap enthält deshalb Methoden entsprechend den Grundoperationen aus Abschnitt 5.2, die für die jeweilige Datenquelle „übersetzt“ werden müssen. Die technische Ausführung der Abfrage, z.B. einer SQL-Query, wird dann durch eine IQueryConnector-Klasse durchgeführt.

Eine besondere Implementierung von IQueryConnector und IActiveMdlMap ist die Klasse InternalDataQueryConnector. Sie führt die Abfragen auf dem internen Datenmodell aus dem Paket model aus. Aus der Sicht von CoreServer, der zentralen Klasse für die Ausführung von Abfragen, ist das Verhalten daher wie das einer IActiveMdlMap. Das bedeutet, dass die Ausführung von Abfragen aus Sicht von CoreServer immer gleich abläuft, unabhängig davon, ob die Abfrage auf dem internen oder einem externen Datenmodell ausgeführt wird. Der ggf. stark unterschiedliche Zeitbedarf ist dabei unkritisch, weil jede Abfrage ohnehin in einem eigenen Thread durchgeführt wird. Der genaue Ablauf eines Dienstaufrufs wird im folgenden Abschnitt 6.2.3 erläutert.

6.2.3. Ablauf eines Dienstaufrufs

Zur Verdeutlichung der Ablaufdynamik wird in diesem Abschnitt die Bearbeitung eines Dienstaufrufs im Klienten und im Server des Merkmaldienstes beschrieben. In Abbildung 6.3 wird der Ablauf als UML-Sequenzdiagramm gezeigt.

Ausgangspunkt des hier dargestellten Dienstaufrufs ist die Situation, dass im Klienten eine gültige Abfrage erstellt und mit einem QUERY-Baustein verbunden wurde. Der QUERY-Baustein ist im Zustand „Leerlauf“ (siehe Abbildung 3.4, Seite 47). Gleichzeitig ist der Merkmaldienst erreichbar und in der Lage die Abfrage zu beantworten.

Sobald am QUERY-Baustein der Wert am Eingang RES von false auf true wechselt, wird die Ausführung der Abfrage angestoßen. Der erste Schritt dazu ist die Aktualisierung des Eingangs Q. Entsprechend Anforderung 9 aus Kapitel 4 (Seite 52) muss die Bedeutung der Ausführungsreihenfolge der Bausteine einer Abfrage dokumentiert und leicht nachvollziehbar sein. Dieser Anforderung wurde in der Implementierung so begegnet, dass grundsätzlich vor der Ausführung einer Abfrage alle zur Abfrage gehörenden Bausteine der Bausteinbibliothek, d.h. alle von opBase erbenden Instanzen, aktualisiert werden. Der Baustein QUERY prüft daher, was für ein Baustein mit dem Eingang Q verbunden ist und traversiert das Bausteinnetzwerk rückwärts solange, bis entweder kein verbundener Baustein mehr gefunden wird oder ein verbundener Baustein nicht

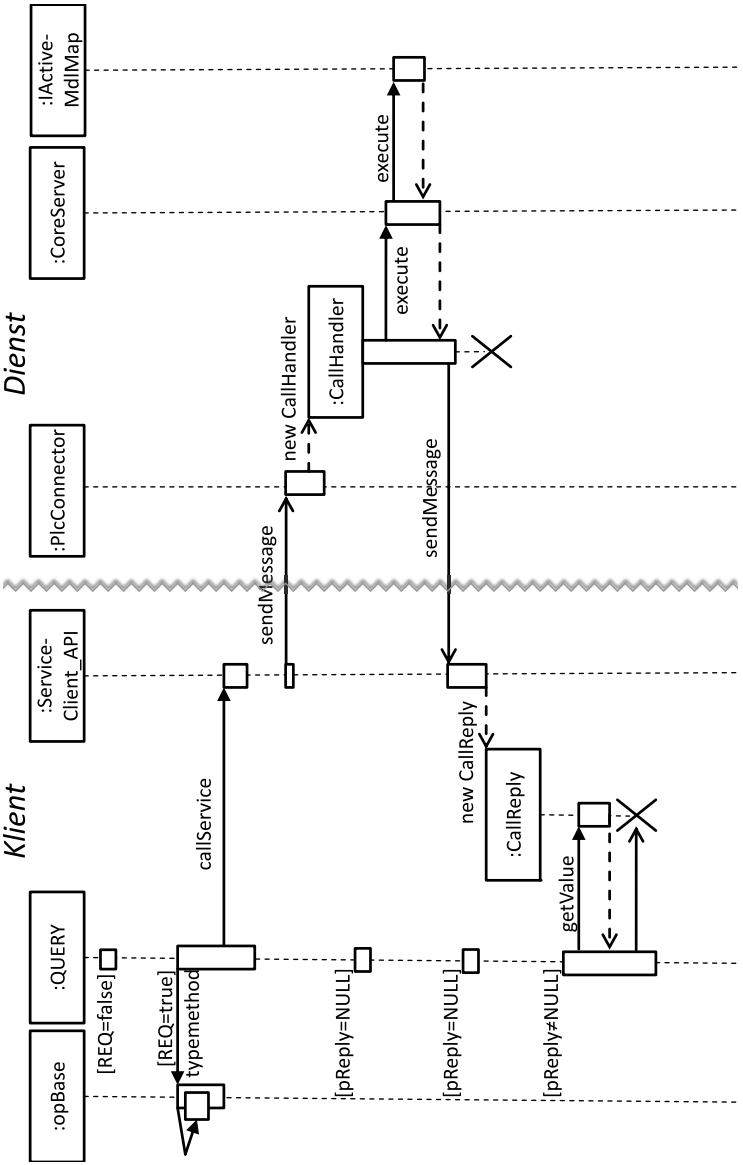

Abbildung 6.3.: Ablauf eines Aufrufs des Merkmaldienstes, dargestellt als UML-Sequenzdiagramm.

von opBase erbt. In Abbildung 6.3 wird das durch die Rekursion bei der Ausführung des opBase-Bausteins gezeigt. Dann werden alle so gefundenen Bausteine der Abfrage durch Ausführung der typemethod in der richtigen Reihenfolge aktualisiert, so dass anschließend der Q-Eingang des QUERY-Bausteins aktuell ist. Dieses Vorgehen hat zwei entscheidende Vorteile: Erstens braucht der Benutzer sich nicht um die Ausführungsreihenfolge der Bausteine zu kümmern, weil sie von den Bausteinen selbst bestimmt wird. Zweitens können so Fehler im Aufbau einer Abfrage, beispielsweise Zyklen, vor der Ausführung festgestellt werden.

Nach der Aktualisierung des Bausteins QUERY erstellt dieser eine Dienstaufrufnachricht und verwendet dann ServiceClient_API zur Durchführung des Dienstaufrufs. Er geht dann in den Zustand „Warten“ über. ServiceClient_API erstellt eine Nachricht für das unterlagerte nachrichtenbasierte Kommunikationssystem und versendet diese zu einem späteren Zeitpunkt. Genauer gesagt wird die Nachricht natürlich vom Kommunikationssystem selbst verschickt, dies wurde aber zur Vereinfachung im Sequenzdiagramm ausgelassen. Die Netzwerkadresse des Dienstes wird dabei aus der aktuellen Konfiguration des Systems, hinterlegt in der Instanz von config (siehe Abbildung 6.1), gelesen. Sie kann daher auch im laufenden Betrieb noch angepasst werden, falls es Änderungen im Netzwerk gibt.

Auf Seite des Merkmaldienstes wird die Nachricht von der Klasse PlcConnector entgegengenommen und geprüft. Anschließend wird ein neuer Thread vom Typ CallHandler erstellt, der für die Ausführung der Abfrage verantwortlich ist. Auf diese Art können mehrere unterschiedlich schnelle Abfragen parallel ohne gegenseitige Behinderung ausgeführt werden. Die CallHandler-Instanz lässt die Abfrage von CoreServer ausführen, der wiederum je nach benötigten Daten die Abfrage an die zuständige Instanz von IActiveMdlMap weiterleitet. Nach vollständiger Ausführung der Abfrage sendet der CallHandler eine Antwortnachricht an den Klienten und der Thread des CallHandlers wird beendet.

Auf Seite des Klienten wird die Nachricht von ServiceClient_API entgegengenommen und ein CallReply-Objekt erstellt. Der QUERY-Baustein prüft zyklisch, ob eine Antwort eingetroffen ist, indem der Wert eines Zeigers geprüft wird. Per Konvention mit ServiceClient_API zeigt dieser Zeiger auf das neu erstellte CallReply-Objekt, sobald es erstellt wurde. Nach Erhalt der Antwort wird diese vom QUERY-Baustein überprüft (Zustand „Prüfen“ in 3.4) und nach erfolgreicher Prüfung werden die Ausgänge NDR und RD gesetzt. Der Baustein geht für einen Zyklus in den Zustand „Daten erhalten“ und dann zurück in „Leerlauf“. Aus Benutzersicht entspricht das Verhalten somit dem des Bausteins READ der DIN EN 61131-5. Außerdem läuft die Abfrage asynchron ab und beeinflusst den Betrieb des Klienten nur durch Erstellen des Dienstaufrufs und Bearbeitung der Antwort, nicht aber durch die inhaltliche Komplexität der Abfrage (vgl. Anforderung 8 und 10, Seite 52).

6.2.4. Administration des Dienstes

Entsprechend Anforderung 11 (siehe Seite 53) sollen Änderungen der Datenanbindung zur Laufzeit unterstützt werden, beispielsweise für den Fall, dass sich die IP-Adresse einer der Datenquellen ändert. In Abschnitt 5.5.2 wird diese Möglichkeit durch Konfigurationsdateien vorgesehen, die für jedes Plug-In einzeln und spezifisch sind. In der

Implementierung des Prototyps wurde diese Idee weitergeführt und die notwendigen technischen Vorkehrungen getroffen.

Im einfachsten Fall muss für das Hinzufügen von Datenquellen zur Laufzeit kein Plug-In nachgeladen werden, weil die neue Datenquelle ein bereits bekanntes Format besitzt. Das Hinzufügen besteht dann nur aus dem Anlegen einer Konfigurationsdatei für die Datenquelle und dem Instanziieren der entsprechenden Objekte im System. Vom Benutzer kann das über die Konsole der Anwendung angestoßen werden. Falls zusätzliche Plug-Ins benötigt werden, müssen diese zunächst geladen werden. Dazu sind die jeweils benötigten Plug-Ins in den Konfigurationsdateien angegeben und es gibt ein spezielles Verzeichnis im Dateisystem, in dem die Implementierung der Plug-Ins (in Form von .jar-Dateien oder .class-Dateien) erwartet wird. Technisch wird das Nachladen von Bibliotheken zur Laufzeit von Java nativ unterstützt. Seitens der Implementierung funktioniert das Nachladen durch die Interfaces im Paket plugin. Die Anwendung stützt sich bei der Verwendung der Plug-Ins allein auf diese Interfaces, d.h. sie müssen nur seitens des Plug-Ins richtig implementiert werden.

Für das Ändern und Entfernen von Datenquellen sehen die Interfaces im Paket plugin jeweils Methoden vor, durch die das jeweilige Plugin die notwendigen Änderungen durchführen kann. Vor der Durchführung von Änderungen überprüft CoreServer, ob derzeit Abfragen auf den betroffenen Datenquellen durchgeführt werden und stellt sicher, dass die Änderungen zwischen zwei Abfragen vorgenommen werden.

6.3. Anwendungsbeispiele

Der Anwendungsbereich der hier beschriebenen Technologie wird durch den Einsatz von Funktionsbausteinen in SPS-basierten Steuerungen und die logische Verwendung von Merkmalinformation definiert. Dadurch ergibt sich ein Spektrum von Anwendungen, das praktisch den gesamten Einsatzbereich von SPS-basierten Steuerungen umfasst. In diesem Abschnitt werden einige Anwendungsszenarien herausgegriffen, die innerhalb von Forschungsprojekten des Lehrstuhls für Prozessleittechnik behandelt wurden. Anhand dieser Beispiele wird gezeigt, wie die Technologie im Sinne der jeweils verfolgten Ziele unterstützend eingesetzt werden kann.

6.3.1. Flexible Programmierung von Werkzeugmaschinen

In einer Forschungskooperation mit dem Werkzeugmaschinenlabor der RWTH Aachen war das Ziel, die Programmierung von Werkzeugmaschinen zu vereinfachen und zu beschleunigen. Besonders bei Routineaufgaben in der Programmierung wie dem Wechsel eines Werkzeugs entsteht ein hoher Programmieraufwand, der durch die Wiederverwendung und geschickte Kombination von einzelnen Programmabschnitten deutlich reduziert werden kann [6]. Der in diesem Forschungsprojekt verfolgte Ansatz ermöglicht die automatische Generierung von Steuerungscode, der nach der Generierung jedoch spezifisch für eine Maschine in einer bestimmten Konfiguration ist.

Durch die in dieser Arbeit eingeführte Technologie lässt sich die Flexibilisierung von Programmen noch weiter vorantreiben, so dass sie ohne Veränderung auch mit unterschiedlichen Maschinen- und Werkzeugkonfigurationen funktionieren, sofern Änderungen nur Werkstück- oder Werkzeugmerkmale betreffen. Damit wird der Aufwand zur

Maschinenprogrammierung weiter verringert, weil eine Neuprogrammierung seltener nötig ist. Beispielsweise müssen für die Programmierung eines Werkzeugwechsels viele Eigenschaften des eingesetzten Werkzeugs berücksichtigt werden, etwa dessen geometrische Merkmale und die maschinenseitig benötigte Werkzeugaufnahme. Anstatt diese Eigenschaften manuell zu übertragen, können sie durch die in Kapitel 5 definierten Funktionsbausteine zur Programmlaufzeit abgefragt werden. Dadurch entfällt der Aufwand beim Übertragen der Merkmalinformation (und potenzielle Fehler), außerdem braucht das Programm nicht erneut auf die SPS geladen zu werden.

Für ein konkretes Beispiel soll angenommen werden, dass ein Werkstück gefräst werden soll und dass dazu unterschiedliche Fräser mit unterschiedlichen Merkmalen, z.B. die Art der Werkzeugaufnahme, zur Verfügung stehen. Der Bediener bzw. Programmierer der Werkzeugmaschine trifft diese Entscheidung. Anstatt die Information zur Werkzeugaufnahme manuell einzutragen (über ein HMI oder direkt im Programmcode), kann die Information vom Programm selbst erfragt werden, indem auf einen eCl@ss-konformen Katalog mit Informationen zu Fräsern zurückgegriffen wird. Die Implementierung wird in Abbildung 6.4 gezeigt. Der gesamte Satz von Werkzeugdaten ist unter der Kennung „Werkz.-Katalog“ hinterlegt und abfragbar. Der linke Baustein erfragt nun denjenigen Schaftfräser (eCl@ss 21-18-06-01), bei dem das Merkmal Artikelbezeichnung (eCl@ss 0173-1#02-AAP805#002) mit dem Inhalt der SPS-Variable „Fraeser“ übereinstimmt. Die Variable wird beispielsweise durch das HMI der Maschine gesetzt. Zu diesem Fräser wird der maschinenseitige Aufnahmetyp (eCl@ss 0173-1#02-BAA763#008) abgefragt, den der Hersteller zusichert, und in die SPS-Variable „AufnTyp“ geschrieben. Die gesamte Abfrage wird durch das Setzen der Variable „StartWWechsel“ ausgeführt.

Ein erheblicher Teil der benötigten Information über Merkmale von Werkzeugen (und ggf. auch Werkstücken) kann auf die beschriebene Art automatisiert im Betrieb erfragt werden. Führt man sich nun vor Augen, dass eCl@ss 9.1 bereits 84 Merkmale für Schaftfräser definiert, wird klar, dass diese Art der Programmierung erheblich zur flexiblen Verwendung von Programmen mit geringem manuellen Aufwand im Betrieb beiträgt.

6.3.2. Überwachung von Erdölpumpen in einer Erdölraffinerie

Zu Beginn dieser Arbeit wurde in Abschnitt 1.1.2 ein Anwendungsszenario geschildert, in dem für die Überwachung von Erdölpumpen die Dichte des geförderten Erdöls benötigt wird. An dieser Stelle wird nun die SPS-seitige Lösung dieser Aufgabenstellung mithilfe der prototypischen Implementierung gezeigt.

Bei der Lösung der Aufgabe wird davon ausgegangen, dass Merkmalinformation aus dem Laborinformations- und Managementsystem (LIMS) und dem Manfacturing Execution System (MES) durch den Merkmaldienst verfügbar sind. Diese Datenquellen können entsprechend durch die Kennungen „LIMS“ und „MES“ beim Merkmaldienst identifiziert werden. Ferner ist bekannt, welche Merkmalträger und Merkmale innerhalb dieser Datensysteme verfügbar sind. Für dieses Beispiel werden zur Identifizierung von Merkmalträgern und Merkmalen sinngemäße Namen benutzt. In der Praxis würden dazu vermutlich eher standardisierte Bezeichner verwendet (siehe Abschnitt 3.1).

Die Abfrage der Dichte des Erdöls besteht logisch gesehen aus zwei Abfrageteilen, weil zunächst der Quelltank aus dem MES und dann die Dichte des Öls in diesem Tank aus dem LIMS erfragt wird. Aus Sicht der SPS wird das in einer Abfrage zusammen-

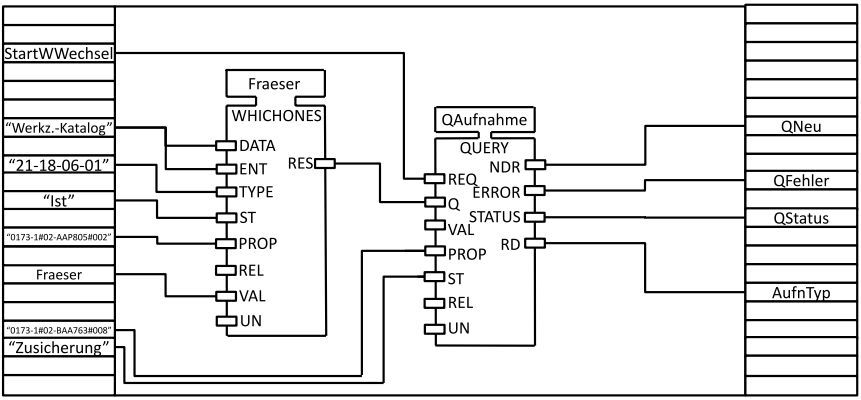

Abbildung 6.4.: Beispielsanwendung der Funktionsbausteinbibliothek. Es wird zunächst der Datensatz zu einem bestimmten Schaftfräser abgefragt und dann daraus die Zusicherung zum maschinenseitigen Aufnahmetyp (codiert in eCl@ss-Kennungen).

gefasst, die dann vom Merkmaldienst in zwei Schritten ausgeführt wird. Die Implementierung dazu wird in Abbildung 6.5 gezeigt. Im Beispiel sei die betroffene Pumpe N7509 und es wird angenommen, dass im MES Produktionsaufträge als Merkmalträger mit Angaben über die verwendeten Ressourcen und den Auftragsstatus als Merkmale vorliegen. Die Funktionsbausteine „Auftraege“ und „AuftrAkt“ finden dann den Auftrag für N7509, der momentan in Bearbeitung ist. Zu beachten ist, dass das Ergebnis dieser Operationen bereits den Merkmalträger mit den Aussagen über ihn zusammenführt. Deswegen kann der folgende Baustein „QuellInfo“ direkt die Aussage über die Quellressource („QuellRess“) zu diesem Auftrag finden. Der „Quelle“ benannte Funktionsbaustein ist dann für die Auswahl des Aussagewerts zuständig, so dass die Kennung des Quelltanks das Ergebnis der Operation VALUE ist.

Die Bausteine im unteren Teil von Abbildung 6.5 bilden den Abfrageteil, der an das LIMS gerichtet ist. Es wird angenommen, dass im LIMS Merkmalträger vom Typ „Probe“ verfügbar sind, und dass diese Merkmalträger ein Merkmal „Entnahmestelle“ besitzen. Entsprechend findet der Baustein „Tankproben“ diejenigen Proben, die aus dem aktuellen Tank entnommen wurden. Unter all diesen Proben wird dann durch einen AGGRALL-Baustein diejenige mit dem aktuellsten Datum herausgesucht. Der Funktionsbaustein „QDichte“ erfragt letztendlich die in dieser Probe gemessene Dichte.

Die gesamte Abfrage wird dann ausgeführt, wenn durch den Wechsel der Variable „PumpeEin“ auf true das Einschalten der Pumpe signalisiert wird. PumpeEin ist der einzige Eingang dieses Diagramms mit variablem Wert, so dass die gesamte, verhältnismäßig komplexe Abfrage sich nach außen relativ einfach darstellt.

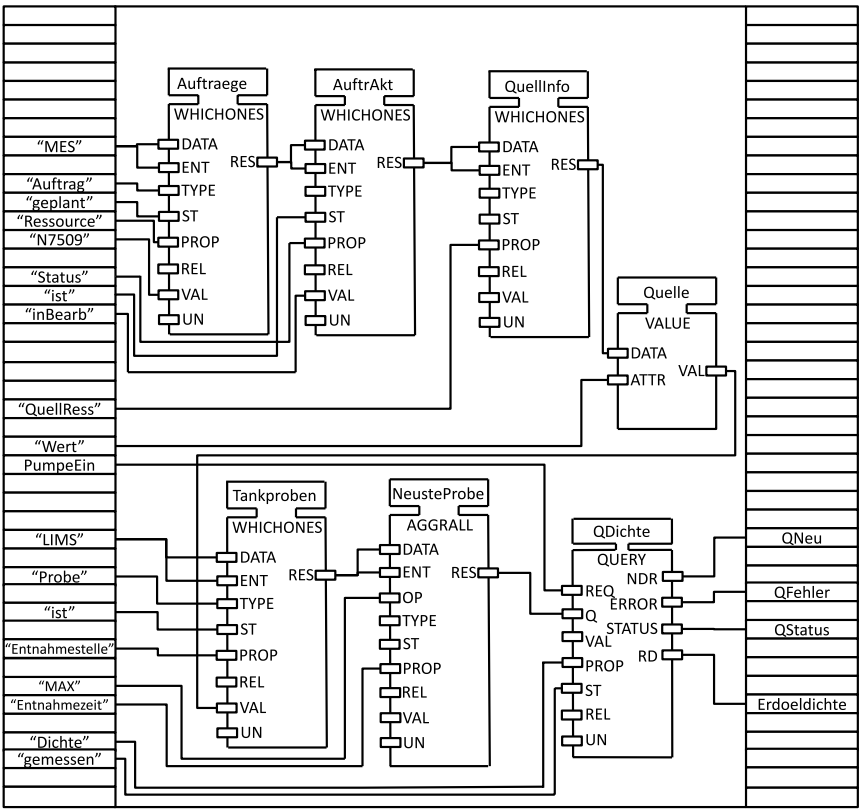

Abbildung 6.5.: Beispielsanwendung der Funktionsbausteinbibliothek. Es wird zunächst der Quelltank des aktuellen Pumpauftrags aus dem MES abgefragt (oben), dann die Dichte des Erdöls darin aus dem LIMS (unten).

7. Diskussion und Ausblick

7.1. Diskussion der Grundidee

Ein wesentlicher Beitrag dieser Arbeit ist die Idee, überhaupt eine Abfragesprache für technische Merkmale zu spezifizieren und zu implementieren. Diese Idee ist natürlich eine Grundvoraussetzung. Hier sind jedoch sowohl die Problemstellung als auch die Lösung verhältnismäßig weit vom aktuellen Stand der Technik entfernt. Es gibt bisher nur wenige Beispiele, in denen Abfragesprachen speziell für bestimmte Anwendungsgebiete definiert wurden (z.B. [44, 56, 68]) und keine für die Abfrage von Merkmalinformation. Folglich muss schon die Grundidee kritisch hinterfragt werden.

Die wichtigste Voraussetzung für die Sinnhaftigkeit der vorgeschlagenen Lösung ist, dass die Kommunikation von Automatisierungssystemen, d.h. hier insbesondere (Soft-) SPSen, zunimmt. Wenn eine SPS nicht an ein Netzwerk angeschlossen ist, oder wenn die SPS oder das Netzwerk nicht für die Kommunikation konfiguriert sind, dann lässt sich diese Lösung nicht umsetzen. Die Verfügbarkeit von netzbasierter Kommunikation ist aber auch die Grundidee der Themen „Industrie 4.0“ und „Industrial Internet“ [1]. Aus dem Grund ist die Wahrscheinlichkeit der zukünftig (noch) höheren Verfügbarkeit von Kommunikationsfähigkeiten äußerst hoch und wird in Fachkreisen auch nicht in Frage gestellt.

Eine größer werdende Menge an kommunikationsfähigen Teilnehmern eines Netzwerks bedeutet fast immer auch eine Vergrößerung der Informationsmenge im Netz. Die einzige Ausnahme davon wäre, wenn die Netzteilnehmer zunehmend weniger oder gar keine Information bereitstellen würden, was aber ein äußerst unwahrscheinliches und widersprüchliches Szenario wäre. Insgesamt ist die Annahme, dass SPSen in Zukunft durch Vernetzung mehr Information zur Verfügung haben werden, realistisch. Technische Lösungen für den Informationszugriff existieren (z.B. [62]).

Die größere Verfügbarkeit von Information wirft die Frage auf, ob der Aufwand zur Nutzung der Information vertretbar gering oder überhaupt zu bewältigen sein wird. In dieser Arbeit wird die These vertreten, dass eine Nutzung der Vernetzung zumindest erhebliche neue Probleme mit sich bringen wird, so dass bald ein Bedarf an neuen Lösungskonzepten besteht. Diese These wird durch drei Beobachtungen gestützt:

- Heute übliche Punkt-zu-Punkt-Verbindungen in einem Netz werden jeweils als Einzelfall angelegt und gewartet.
- Die Anzahl möglicher Punkt-zu-Punkt-Verbindungen in einem Netz wächst quadratisch mit der Anzahl der Teilnehmer.
- Die Informationsverfügbarkeit führt zu neuen Ideen zur Informationsnutzung, die wiederum neue Information erzeugen können und neue Nutzungsmöglichkeiten eröffnen.

Daher wird die Korrektheit der These als realistisch eingestuft. Für die Verringerung der zu erwartenden Aufwände gibt es mehrere denkbare Möglichkeiten, beispielsweise könnte auch versucht werden, die Verwaltung von Punkt-zu-Punkt-Verbindungen zu automatisieren. Die hier vertretene Lösung, den Informationszugriff über einen zentralen Dienst zu realisieren, ist daher nur eine Möglichkeit. Für diese Möglichkeit spricht in erster Linie die Erfahrung aus dem Internet, in dem sich beispielsweise Domain Name Server und Suchmaschinen als zentrale Zugriffspunkte für verteilte Information etabliert haben.

Die nächste wichtige Frage ist, warum der Informationsaustausch auf Basis von Merkmalen geschehen soll. Ein wichtiges Argument dafür ist, dass ein zentraler Dienst zur Informationsabfrage nur dann funktionieren kann, wenn es Konventionen über die Semantik der Daten gibt. Wenn die Semantik je nach ursprünglicher Informationsquelle grundverschieden ist, kann der Dienstnutzer nicht mehr von dieser Informationsquelle abstrahieren und ein wesentlicher Vorteil des Dienstes wird verloren. Dies ist ein wichtiger Unterschied zu Suchmaschinen im Internet, bei denen es keine Zusicherungen über die Semantik der gefundenen Information gibt. Für den hier diskutierten Dienst stellt sich also die Frage nach der semantischen Basis. Technische Merkmale sind eine besonders einfache und weit anwendbare semantische Basis und es „erscheint [...] möglich, mit standardisierten Merkmalmodellen einen Interoperationsgrad zu erreichen, der für viele Anforderungen ausreichend ist“ [29]. Die Wahl von Merkmalen als Basis lässt sich also nicht durch harte Fakten begründen, ist aber in Anbetracht des Anwendungsbereichs eine sinnvolle Wahl.

Die Spezifikation einer Abfragesprache ist dann eine direkte Konsequenz aus möglichen Anwendungsszenarien. Die Möglichkeit der Abfrage eines bestimmten Aussagewerts über einen bekannten Merkmalträger ist in vielen Anwendungen nicht ausreichend. Stattdessen wird oft eine Information gesucht, für die sehr spezielle und komplexe Bedingungen gelten (wie im Beispiel in Abschnitt 6.3). Die Verwendung einer vollständigen Abfragesprache anstelle einfacher Leseoperationen ist daher eine praktische Notwendigkeit.

Insgesamt beruht die Argumentation, warum eine Lösung dieser Art sinnvoll und sogar notwendig ist, auf plausiblen und wahrscheinlichen Annahmen. Ein Kritikpunkt ist jedoch, dass Entwicklungen vorweggenommen werden und letztlich ein Problem gelöst wird, das heute noch nicht in einer relevanten Größe existiert. Beim tatsächlichen Auftreten des Problems können bereits andere Technologien und Standards verfügbar sein, die in dieser Lösung nicht berücksichtigt sind. Letztlich erscheint es dennoch wahrscheinlich, dass Grundideen dieser Arbeit verwendet werden können.

7.2. Diskussion der Abfragesprache

Durch die Verwendung relationaler Algebra als Grundlage besteht eine gesicherte, anerkannte und der Anwendung gerechte Basis. Die fehlende Funktionalität der relationalen Algebra wurde passend ergänzt. Von der theoretischen Seite ist die Abfragesprache somit gut begründet. In ähnlichen Arbeiten zur Erstellung spezieller Abfragesprachen wurde ebenfalls dieser Weg bestritten (z.B. [56, 68]).

Eine andere Frage ist der tatsächliche praktische Nutzen. Das hier vollzogene Vorgehen ist systematisch: Die relevanten technischen Grundlagen im industriellen Umfeld

wurden betrachtet und daraus Anforderungen abgeleitet, die von der Lösung erfüllt werden. Es verbleibt aber der Kritikpunkt, dass die Abfragesprache mehr durch Technik und Theorie als durch empirisch nachgewiesenen Nutzen motiviert ist: In Abschnitt 5.3 wurden Operationen der Abfragesprachen unter Nennung von praktischen Beispielen definiert. Ob diese Anwendungsfälle realistisch sind, ob der Einsatz der Abfragen einen praktischen Gewinn darstellt und ob es nicht unerfüllbare Nebenbedingungen gibt, wurde nicht analysiert. Das ist natürlich dadurch zu erklären, dass es derzeit keine vergleichbare Technologie und Erfahrungen damit gibt. Letztlich ist die praktische Anwendbarkeit der Abfragesprache aber nur eine begründete Vermutung, deren Beweis noch ausbleibt. Dieser Kritikpunkt wiegt weniger schwer, weil die zukünftige Spezifikation zusätzlicher Operationen nicht ausgeschlossen ist und sogar erwartet wird.

7.3. Diskussion der Integration

Die beschriebene Integration der Technologie stützt sich auf existierende Normen und Standards. Eine Untersuchung der in Unternehmen heute vorhandenen Technologien hat jedoch nicht stattgefunden. Beispielsweise fehlt der Beweis dafür, dass die in Kapitel 3.3 beschriebenen Software-Systeme tatsächlich geeignete Datenschnittstellen anbieten. In dieser Hinsicht stützt sich die Arbeit auf die Annahme, dass real verwendete Technologien nicht „zu weit“ von diesen Normen und Standards entfernt sind.

In Abschnitt 3.1 wurde diskutiert, welche Merkmalmodelle und Katalogsysteme heute verfügbar sind. Die Verfügbarkeit von systematischen und großen Merkmaldefinitionen ist daher nicht fraglich. Ob diese Merkmaldefinitionen in der industriellen Praxis tatsächlich Anwendung finden ist damit aber noch nicht gesagt. Für die Integration der hier vorgeschlagenen Technologie ist der Einsatz einheitlicher Merkmaldefinitionen aber eine zwingende Voraussetzung und macht bei Nichterfüllung einen erfolgreichen Einsatz unmöglich. Das ist aber nicht nur ein Problem dieser konkreten Technologie und wird ohnehin eine zukünftige Herausforderung der Automatisierungstechnik sein.

In der prototypischen Implementierung wurden Daten im BMEcat-Format integriert. Diese Integration war ohne Probleme durchführbar, was jedoch kein Beweis dafür ist, dass auch andere existierende Datenformate und Informationssysteme problemlos angebunden werden können. Die universelle Gültigkeit des in Kapitel 2.1 vorgestellten Merkmalmodells wird vorausgesetzt und ist eine kritische Vorbedingung für das Funktionieren der hier beschriebenen Technologie. Neben dieser Frage der grundsätzlichen Abbildbarkeit von existierenden Datenformaten spielt in der Praxis natürlich auch die Schnelligkeit der Datenverarbeitung eine Rolle, denn jede Abfrage muss innerhalb eines gewissen Zeitfensters beantwortet werden (selbst wenn es keine harten Echtzeitbedingungen gibt). Dieses Problem wird von dieser Arbeit absichtlich nicht behandelt, weil es als Problem der Implementierung im Einzelfall gesehen wird und weil Fragestellungen dieser Art in den fachlichen Bereich der Informatik und der Softwaretechnik fallen, wo sie bereits behandelt werden.

Die Implementierung von Abfragen durch Funktionsbausteine hat zwei wesentliche Gründe: Erstes werden in der IEC 61131-5 ebenfalls Funktionsbausteine für Kommunikationsaufgaben verwendet und diese Spezifikation kann teilweise übernommen werden. Zweitens sind Funktionsbausteine genau wie Abfragesprachen eine deklarative Programmiersprache, so dass diese Paradigmen gut zusammenpassen. Letztlich wäre

es aber alternativ möglich, Abfragen wie bei Abfragesprachen wie SQL textuell zu implementieren und ggf. nur die Ausführung durch einen QUERY-Funktionsbaustein vorzunehmen. Eine Untersuchung, welche der Vorgehensweisen komfortabler und damit auch besser akzeptiert wäre, hat in dieser Arbeit nicht stattgefunden.

7.4. Ausblick

Für die erfolgreiche Integration und Anwendung des Konzepts werden noch einige Arbeiten notwendig sein. Dies sind im Einzelnen:

- Existierende, genormte bzw. standardisierte Merkmale und Schnittstellen müssen angewendet werden. Es ist heute nicht das Problem, dass solche Standards nicht vorhanden wären, sondern dass sie noch nicht breit eingesetzt werden und dass die Anwendung in Details oft nicht einheitlich ist. Solche kleinen Unterschiede waren bisher wegen der mehr manuellen und lokalen Datenverarbeitung ggf. nicht relevant, würden aber als Probleme auftreten. Der positive Aspekt daran ist, dass solche Inkonsistenzen schneller als bisher erkannt würden und durch den Nutzen auch eine Motivation zur Vereinheitlichung besteht.

- Die von einzelnen Systemen veröffentlichte Merkmalinformation muss definiert und verwaltet werden. Letztlich kann die Verfügbarkeit von Information auch zu einem Problem werden, wenn sich andere Systeme auf die bestehende Verfügbarkeit verlassen und ggf. sogar weitere Information daraus folgern. Daher müssen organisatorische Strukturen eingeführt werden, durch die die Rechte zur Veröffentlichung und zum Informationszugriff verwaltet und technisch durchgesetzt werden. Wenn durch die hier vorgestellte Technologie auch schreibende Zugriffe ermöglicht werden sollen, sind entsprechend strengere Maßnahmen notwendig.

- Die vorhandenen Kommunikationsnetze müssen auf ihre Eignung hin überprüft und evtl. angepasst werden. Dies betrifft beispielsweise die Verbindung einer großen Anzahl von Endgeräten, Gewährleistung einer ausreichend schnellen Anbindung des Dienstes für Merkmalabfragen und das Ermöglichen der Kommunikation über Subnetze hinweg (z.B. durch entsprechende Konfiguration von Firewalls).

- Der Merkmaldienst muss für den Betrieb in der industriellen Praxis implementiert werden. Im Rahmen dieser Arbeit wurde eine prototypische Implementierung erstellt, deren Zweck jedoch allein das Aufzeigen der Umsetzbarkeit des Konzepts ist. Für den Praxisbetrieb spielen die Skalierbarkeit des Dienstes, die Zusicherung einer Dienstqualität und die Verfügbarkeit der notwendigen Plug-Ins zur Datenanbindung eine wichtige Rolle. Diese Aspekte machen die Implementierung eines praxisgerechten Merkmaldienstes zu einer umfangreichen Aufgabe, für die der Prototyp und die Pläne zur Systemarchitektur nur einige Grundlagen sind. Hier kann ggf. auf vorhandene Technologien aus den Gebieten „Enterprise Information Integration“ und „Data Warehouses“ zurückgegriffen werden.

- Der letzte (aber entscheidende) Schritt ist natürlich die Entwicklung von Anwendungen, die die Technologie nutzen. Experten rechnen damit, dass durch die Mög-

lichkeiten der Vernetzung industrieller Produktionsanlagen zunächst nur quantitative Verbesserungen erzielt werden, beispielsweise bessere Produktqualitäten, geringere Kosten und mehr Produktvarianten. Erst darauf folgt die Entwicklung neuer Geschäftsmodelle, die ohne Vernetzung nicht möglich gewesen wären [77]. Mit der hier vorgestellten Technologie wird es sich ähnlich verhalten. Zu den heute offensichtlichen Einsatzmöglichkeiten werden zusätzliche, heute noch unbekannte Anwendungen hinzukommen. Die Nutzung der Technologie wird selbst zur Entwicklung dieser Anwendungen beitragen.

A. Spezifikation der Funktionsbausteine

Die folgenden Zeilen spezifizieren die in dieser Arbeit definierten Funktionsbausteine in Form einer Modelldatei für das OV-Laufzeitsystem.

```
/**

        PropertyInfos.ovm

        This Library provides function blocks for building and
           executing queries on property data.
        In order to work, a respective property service needs to be
           running and the following parameters need to be configured
           in the server's config file:
        PropertyService-IP=<IP-Address>
        PropertyService-Port=<Port number>
        PropertyService-Retry-Connect=<Max. number of connection
           attempts>

*/
#include "ov.ovm"
#include "fb.ovm"
#include "ServiceClient.ovm"

LIBRARY PropertyInfos
        VERSION   = "V0.1 (23-Oct-2015)";
        AUTHOR    = "David Kampert";
        COMMENT   = "Library for request of property information from
           remote systems.";

        /**
                The Query block. Executes the queries.
        */
        CLASS QUERY : CLASS fb/functionblock
                IS_INSTANTIABLE;
                VARIABLES
                        REQ : BOOL HAS_SET_ACCESSOR FLAGS = "i"
                           COMMENT = "Reqest. The query is executed if
                            the value of this imput changes to true."
                           INITIALVALUE = FALSE;
                        Q : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                           COMMENT = "Query. Input port of the query."
                            INITIALVALUE = "";
                        VAL : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                           COMMENT = "Value. Input port to identify
```

```
                    the value from the query result that will
                    be written to RD." INITIALVALUE = "";
                PROP : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                    COMMENT = "Property. If the value of PROP
                    is not empty, the query will return the
                    value of a specific statement on this
                    property, provided that Q represents a set
                    of property carriers or property carrier
                    types. The statement is specified by the
                    inputs ST, REL, VAL and UN." INITIALVALUE =
                     "";
                ST: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Statement Type. The type of the
                    queried statement on PROP." INITIALVALUE =
                    "";
                REL: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Relation. The relation type of
                    the queried statement on PROP."
                    INITIALVALUE = "";
                UN: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Unit. The unit of the queried
                    statement on PROP." INITIALVALUE = "";
                NDR : BOOL HAS_GET_ACCESSOR FLAGS = "o"
                    COMMENT = "New Data Received. Flag that
                    indicates the advent of a result. Is reset
                    after one cycle." INITIALVALUE = FALSE;
                ERROR : BOOL HAS_GET_ACCESSOR FLAGS = "o"
                    COMMENT = "Error. Indicates of any error
                    occured during execution of the query.
                    False, if no error occured." INITIALVALUE =
                     FALSE;
                STATUS : INT HAS_GET_ACCESSOR FLAGS = "o"
                    COMMENT = "Status. Output of the error code
                     according to IEC 61131-5, if any error
                    occured. 0, if no error occured."
                    INITIALVALUE = 0;
                RES : ANY HAS_GET_ACCESSOR FLAGS  = "o"
                    COMMENT = "Result.";
                state : INT FLAGS = "n" COMMENT = "Internal
                    state of execution." INITIALVALUE = 0;
                pReply : C_TYPE <OV_INSTPTR> COMMENT = "
                    Pointer to an expexted call answer.";
                try : INT FLAGS = "n" COMMENT = "Number of
                    cennection attempts." INITIALVALUE = 0;
        END_VARIABLES;
        OPERATIONS
                constructor : C_FUNCTION <OV_FNC_CONSTRUCTOR>;
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;
```

```
CLASS OpBase : CLASS fb/functionblock
        VARIABLES
                RES : STRING HAS_GET_ACCESSOR FLAGS  = "o"
                   COMMENT = "Result. The operation result."
                   INITIALVALUE = "";
        END_VARIABLES;
END_CLASS;

/**
        The Reduce block. Will reduce the provided data to the
            provided attributes. Useful to 'cut off'
           unnecessary information.
*/
CLASS REDUCE : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                ATTR : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Attributes. The attributes to
                   reduce to, i.e. 'what will be left'."
                   INITIALVALUE = "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The Value block. Gets the value of a spcified
           attribute.
*/
CLASS VALUE : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                ATTR : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Attribute. The result is the
                   value of this attribute'." INITIALVALUE =
                   "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;
```

```
/**
        The Choose block. Will reduce the provided data to
           those elements that fulfil the provided condition,
           e.g. a minimum age.
*/
CLASS CHOOSE : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                COND : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Condition. The condition must
                   evaluate to a Boolean value, but may
                   contain arithmetic computations and String
                   comparisons ($== for equals, $!= for not
                   equals). Variables may be referred to by
                   the syntax <type>:<attribute>, e.g.
                   STATEMENT:VALUE." INITIALVALUE = "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The Combine block. The result is the carthesian
           product of the inputs, i.e. all possible
           combinations of the elements in DATA1 and DATA2.
*/
CLASS COMBINE : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA1 : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                DATA2 : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The Union block. The result is the union of the
           elements in DATA1 and DATA2. The elements have to
           possess the same attributes.
*/
```

```
CLASS UNION : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA1 : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                DATA2 : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The Remove block. Removes the elements in DATA2 from
           DATA1, i.e. RES = DATA1 - DATA2.
*/
CLASS REMOVE : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA1 : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                DATA2 : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The Rename block. Renames attributes.
*/
CLASS RENAME : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                OLD : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Old name of the attribute."
                   INITIALVALUE = "";
                NEW : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "New name of the attribute."
                   INITIALVALUE = "";
        END_VARIABLES;
```

```
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The TypeOf block. Gets the types of the provided
           elements, i.e. the property carrier types or
           property types. Both kinds of elements may be
           provided.
*/
CLASS TYPEOF : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                SUB : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Subordinate. The types of these
                   objects are returned." INITIALVALUE = "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The Aggregate All block. Aggregates the values of
           statements or results of other operations by a
           given operation.
*/
CLASS AGGRALL : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to.
                   Must either be statements or results of
                   other operations." INITIALVALUE = "";
                ENT : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Entities. The set of entities (
                   property carriers or property carrier types
                   ) that the aggregation is applied to. If
                   this input is empty, the concents of DATA
                   will be used. If this input is not empty,
                   the aggregation will only be applied to the
                    statements on entities in ENT that match
                   the citeria defined by the other inputs."
                   INITIALVALUE = "";
                OP : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Operation. The operation to
```

```
                    apply. Supported: +,-,*,/,MAX,MIN,NUM."
                    INITIALVALUE = "";
                TYPE: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "The property carrier type in ENT
                     to match." INITIALVALUE = "";
                ST: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Statement Type. The statement
                    type to match." INITIALVALUE = "";
                PROP: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Property. The property in ENT to
                     match." INITIALVALUE = "";
                REL: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Relation. The relation type to
                    match." INITIALVALUE = "";
                VAL: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Value. The value into match."
                    INITIALVALUE = "";
                UN: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Unit. The unit to match."
                    INITIALVALUE = "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The Aggregate Each block. Aggregates the values of the
            provided attributes in each provided element by a
           given operation.
*/
CLASS AGGREACH : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Data to apply the operation to."
                     INITIALVALUE = "";
                ENT : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                    COMMENT = "Entities. The set of entities (
                    property carriers or property carrier types
                    ) that the aggregation is applied to. If
                    this input is empty, the concents of DATA
                    will be used. If this input is not empty,
                    the aggregation will only be applied to the
                     statements on entities in ENT that match
                    the citeria defined by the other inputs."
                    INITIALVALUE = "";
                ATTR : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                    COMMENT = "Attributes. Comma-separated list
                     of attribute names. The operation will be
```

```
                    applied to the attributes in the provided
                    order." INITIALVALUE = "";
                OP : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                    COMMENT = "Operation. The operation to
                    apply. Supported: +,-,*,/,MAX,MIN,NUM."
                    INITIALVALUE = "";
                TYPE: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "The property carrier type in ENT
                     to match." INITIALVALUE = "";
                ST: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Statement Type. The statement
                    type to match." INITIALVALUE = "";
                PROP: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Property. The property to match
                    ." INITIALVALUE = "";
                REL: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Relation. The relation type  to
                    match." INITIALVALUE = "";
                VAL: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Value. The value to match."
                    INITIALVALUE = "";
                UN: STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Unit. The unit to match."
                    INITIALVALUE = "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The Join Statements block. Joins entities (property
           carriers or property carrier types) with all
           statements from DATA that affect them. The result
           is a list that has as many entries as relevant
           statements, where each entry contains information
           from the property carrier (type) and the statement.
*/
CLASS JOINSTMT : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                    COMMENT = "Data to apply the operation to."
                     INITIALVALUE = "";
                ENT : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                    COMMENT = "Entities. The set of entities (
                    property carriers or property carrier types
                    ) that should be joined with the respective
                     statements." INITIALVALUE = "";
        END_VARIABLES;
```

```
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The Has A Property block. Returns those entities (
           property carriers or property carrier types) that
           possess a given property.
*/
CLASS HASAPROP : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                ENT : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Entities. The set of entities (
                   property carriers or property carrier types
                   ) are searched for a property."
                   INITIALVALUE = "";
                PROP: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "The property to search."
                   INITIALVALUE = "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The Is Of Type block. Returns those entities (property
            carriers or property carrier types) that are of a
           given property carrier type.
*/
CLASS ISOFTYPE : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                ENT : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Entities. The set of entities (
                   property carriers or property carrier types
                   ) to check for a type. Any data that does
                   not describe property carriers or property
                   carriert types within this set will be
                   ignored." INITIALVALUE = "";
                TYPE: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "The property carrier type to
```

```
                    search." INITIALVALUE = "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
        The Whichones block. Returns those entities (property
           carriers or property carrier types) for which a
           statement exists, that matches given criteria.
*/
CLASS WHICHONES : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                ENT : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Entities. The set of entities (
                   property carriers or property carrier types
                   ) to check for the criteria. Any data that
                   does not describe property carriers or
                   property carriert types within this set
                   will be ignored." INITIALVALUE = "";
                TYPE: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "The property carrier type to
                   match." INITIALVALUE = "";
                ST: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Statement Type. The statement
                   type to match." INITIALVALUE = "";
                PROP: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Property. The property to match
                   ." INITIALVALUE = "";
                REL: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Relation. The relation type to
                   match." INITIALVALUE = "";
                VAL: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Value. The value to match."
                   INITIALVALUE = "";
                UN: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Unit. The unit to match."
                   INITIALVALUE = "";
        END_VARIABLES;
        OPERATIONS
                typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
        END_OPERATIONS;
END_CLASS;

/**
```

```
        The Matching block. Returns those property carriers
           from ENT1 and ENT2 for which anoter entity in ENT2
           (rep. ENT1) exists that matches in the indicated
           criteria.
*/
CLASS MATCHING : CLASS PropertyInfos/OpBase
        IS_INSTANTIABLE;
        VARIABLES
                DATA : STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Data to apply the operation to."
                    INITIALVALUE = "";
                ENT1 : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Entities. The set of entities (
                   property carriers or property carrier types
                   ) to compare with entities from ENT2."
                   INITIALVALUE = "";
                ENT2 : STRING HAS_SET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Entities. The set of entities (
                   property carriers or property carrier types
                   ) to compare with entities from ENT2."
                   INITIALVALUE = "";
                MID: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Match Id. Indicates if the id
                   should be matched (TRUE of FALSE)."
                   INITIALVALUE = "";
                MTY: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Match Type. Indicates if the
                   property carrier type should be matched (
                   TRUE of FALSE)." INITIALVALUE = "";
                MPR: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Match Property. Indicates if the
                    property type should be matched (TRUE of
                   FALSE)." INITIALVALUE = "";
                MST: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Match Statement. Indicates if
                   the statement type should be matched (TRUE
                   of FALSE)." INITIALVALUE = "";
                MRE: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Match Relation. Indicates if the
                    statement relation should be matched (TRUE
                    of FALSE)." INITIALVALUE = "";
                MUN: STRING HAS_SET_ACCESSOR FLAGS = "i"
                   COMMENT = "Match Unit. Indicates if the
                   statement unit should be matched (TRUE of
                   FALSE)." INITIALVALUE = "";
                REL : STRING HAS_GET_ACCESSOR FLAGS  = "i"
                   COMMENT = "Relation. That the statement
                   values should be in (=,!=,<,<=,>,>=)."
                   INITIALVALUE = "";
        END_VARIABLES;
```

```
                OPERATIONS
                        typemethod  : C_FUNCTION <FB_FNC_TYPEMETHOD>;
                END_OPERATIONS;
        END_CLASS;

        /**
                Only for internal configuration management.
        */
        CLASS Config : CLASS ov/object
                IS_INSTANTIABLE;
                VARIABLES
                        ServiceIP : STRING COMMENT = "IP address of
                           the service's server." INITIALVALUE = "";
                        ServicePort : STRING COMMENT = "Port on the
                           service's server." INITIALVALUE = "";
                        RetryConnect : INT COMMENT = "Number of
                           connection attempts. Connection fails after
                            failing the specified number is reached."
                           INITIALVALUE = 1;
                END_VARIABLES;
                OPERATIONS
                END_OPERATIONS;
        END_CLASS;
END_LIBRARY;
```

B. Verhalten des Bausteins QUERY

Die folgende C-Funktion enthält die Ausführungslogik des Bausteins QUERY. Unterfunktionen sind aufgrund des Umfangs nicht abgedruckt.

```
/*
        Main function of function block QUERY.
*/
OV_DLLFNCEXPORT void PropertyInfos_QUERY_typemethod(
        OV_INSTPTR_fb_functionblock     pfb,
        OV_TIME *pltc
) {
    OV_INSTPTR_PropertyInfos_QUERY pinst = Ov_StaticPtrCast(
       PropertyInfos_QUERY, pfb);
    OV_INSTPTR_PropertyInfos_Config pConfig;
    OV_INT_VEC blockStack = {0, NULL};
    OV_RESULT result = OV_ERR_OK;

    switch (pinst->v_state){
        case STATE_IDLE:
                pinst->v_NDR = FALSE;
                pinst->v_ERROR = FALSE;
                return;

        case STATE_REQ:
                pinst->v_REQ = FALSE;

                // Update the inputs. Triggers a backwards search that
                    will update all connected blocks of type OpBase.
                Ov_SetDynamicVectorLength(&blockStack, 0, INT);
                result = updateQueryInput(&blockStack, pinst, NULL);
                Ov_SetDynamicVectorLength(&blockStack, 0, INT);

                // Check the inputs.
                        if (result != OV_ERR_OK || !checkInputs(pinst)
                           ){
                                pinst->v_state = STATE_ERROR;
                                pinst->v_ERROR = TRUE;
                                pinst->v_STATUS =
                                   IEC_STATUS_INPUT_ERROR;
                                return;
                        }

                // Do the call.
                pinst->v_pReply = NULL;
```

```
        if (!executeRequest(pinst, (
           OV_INSTPTR_ServiceClient_CallReply*)&pinst->
           v_pReply, getTmpDomain(pinst))){
                pinst->v_state = STATE_ERROR;
                        pinst->v_ERROR = TRUE;
                        pinst->v_STATUS = IEC_STATUS_NO_COMM;
                        return;
        }

        // Reset the output.
        ov_string_setvalue(&pinst->v_RES.value.valueunion.
           val_string, NULL);
        // Now wait.
        pinst->v_state = STATE_WAITING;
        pinst->v_STATUS = IEC_STATUS_BUSY;
        return;

case STATE_WAITING:
        pinst->v_try++;
        if (pinst->v_pReply == NULL){
                pConfig = (OV_INSTPTR_PropertyInfos_Config)
                   ov_path_getobjectpointer(CONFIGPATH, 0);
                if (pinst->v_try > pConfig->v_RetryConnect){
                        // Max attempts reached.
                        pinst->v_state = STATE_ERROR;
                        pinst->v_STATUS = IEC_STATUS_NO_COMM;
                }
                break;
        }
        // We have an answer if this line is reached.
        result = OV_ERR_GENERIC;
        result = handleReply(pinst);
        if (!Ov_OK(result)){
                pinst->v_state = STATE_ERROR;
                pinst->v_STATUS = IEC_STATUS_DATA_ERROR;
                break;
        }
        pinst->v_state = STATE_NEWDATA;
        pinst->v_STATUS = IEC_STATUS_OK;
        break;

case STATE_NEWDATA:
        pinst->v_NDR = TRUE;
        pinst->v_state = STATE_IDLE;
        pinst->v_try = 0;
        deleteTmpDomain(pinst);
        break;

case STATE_ERROR:
        pinst->v_ERROR = TRUE;
```

```
                pinst->v_state = STATE_IDLE;
                pinst->v_try = 0;
                deleteTmpDomain(pinst);
                break;
    }
    return;
}
```

C. Abkürzungsverzeichnis

ACPLT	Aachener Prozessleittechnik
AS	Ablaufsprache
AT	Automatisierungstechnik
AWL	Anweisungsliste
CAD	Computer Aided Design
FB	Funktionsbaustein (Eigenname einer Bibliothek)
FBS	Funktionsbausteinsprache
IT	Informationstechnologie
KOP	Kontaktplan
KS	Kommunikationssystem (Kommunikationsprotokoll und Eigenname einer Bibliothek)
LIMS	Labor-Informations und Management-System
MES	Manufacturing Execution System
OPC DA	Object Linking and Embedding for Process Control Data Access (Eigenname)
OPC UA	Object Linking and Embedding for Process Control Unified Architecture (Eigenname)
OV	Objektverwaltung (Eigenname eines Laufzeitsystems)
PC	Personal Computer
R&I~	Rohrleitungs- und Instrumenten~
SPS	Speicherprogrammierbare Steuerung
SQL	Structured Query Language
ST	Strukturierter Text
UML	Unified Modeling Language

Literaturverzeichnis

[1] acatech – Deutsche Akademie der Technikwissenschaften (2013). Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0.

[2] Ahrens, W. (2010). Eine Gegenüberstellung von VDI/VDE 3682, PROLIST, eCl@ss. *atp – Automatisierungstechnische Praxis 9/2010*, 32–45.

[3] Ahrens, W. und M. Polke (1994). *Prozeßleittechnik*, Chapter 2, S. 21–90. Oldenbourg Verlag.

[4] American National Standards Institute (2011, Mai). ANSI/INCITS/ISO/IEC 9899:2011: Information technology - Programming language - C.

[5] Barth, M. und A. Fay (2010, Nov). Efficient use of data exchange formats in engineering projects by means of language integrated queries – Engineers LINQ to XML. In *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, S. 1335–1340.

[6] Brecher, C., W. Herfs, D. Behnen und J. Flender (2015). Planungsunterstützte Programmierung von Steuerungssystemen. In *Automation 2015: Benefits of Change - the Future of Automation*, S. 1055–1066. VDI-Verlag.

[7] Chamberlin, D. D. und R. F. Boyce (1974). SEQUEL: A Structured English Query Language. In *Proceedings of 1974 ACM-SIGMOD Workshop on Data Description, Access and Control, Ann Arbor, Michigan, May 1-3, 1974, 2 Volumes*, S. 249–264.

[8] Chen, P. (1976). The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems 1*, 9–36.

[9] Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Commun. ACM 13*(6), 377–387.

[10] Codd, E. F. (1972). Relational Completeness of Data Base Sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California*.

[11] Connolly, T., C. Begg und A. Strachan (2002). *Datenbanksysteme*. Addison-Wesley.

[12] Defense Advanced Research Projects Agency (1981a, September). RFC 791: Internet Protocol. URL `https://tools.ietf.org/html/rfc791`, (besucht am 09.12.2015).

[13] Defense Advanced Research Projects Agency (1981b, September). RFC 793: Transmission Control Protocol. URL `https://tools.ietf.org/html/rfc793`, (besucht am 09.12.2015).

[14] Deutsches Institut für Normung. DIN EN 61512: Chargenorientierte Fahrweise (alle Teile).

[15] Deutsches Institut für Normung. DIN EN 62541: OPC Unified Architecture (alle Teile).

[16] Deutsches Institut für Normung. DIN EN ISO 80000: Größen und Einheiten (alle Teile).

[17] Deutsches Institut für Normung (2001, November). DIN EN 61131-5:2001: Speicherprogrammierbare Steuerungen – Teil 5: Kommunikation.

[18] Deutsches Institut für Normung (2004, März). DIN EN 61131: Speicherprogrammierbare Steuerungen (alle Teile).

[19] Deutsches Institut für Normung (2004, Dezember). DIN EN 61360-1: Genormte Datenelementtypen mit Klassifikationsschema für elektrische Bauteile – Teil 1: Definitionen.

[20] Deutsches Institut für Normung (2011, September). DIN ISO 17356:2011: Zustandsüberwachung und -diagnostik von Maschinen – Allgemeine Anleitungen.

[21] Deutsches Institut für Normung (2014a, Juni). DIN EN 61131-3:2014: Speicherprogrammierbare Steuerungen – Teil 3: Programmiersprachen.

[22] Deutsches Institut für Normung (2014b, September). DIN EN 61499: Funktionsbausteine für industrielle Leitsysteme (alle Teile).

[23] Deutsches Institut für Normung. DIN 4002: Merkmale und Geltungsbereiche zum Produktdatenaustausch (alle Teile).

[24] Deutsches Institut für Normung (2014). *DIN SPEC 40912: Kernmodelle - Beschreibung und Beispiele*. Beuth Verlag.

[25] Diedrich, C., M. Meyer, L. Evertz und W. Schäfer (2014). Dienste in der Automatisierungstechnik - Automatisierungsgeräte werden I40-Komponenten. *atp edition 12/2014*.

[26] eCl@ss e.V. URL `http://www.eclass.de`, (besucht am 09.12.2015).

[27] Eibl, M., D. Westphal, P. Zgorzelski, U. Kaptein und H.-J. Rudolf (2000). eCl@ss – ein Werkzeug zur Unterstützung der Prozesse im eCommerce, der Materialwirtschaft und der Anlagendokumentation, bezogen auf das PLT-Gewerk. *atp – Automatisierungstechnische Praxis 10/2000*.

[28] Enste, U. (2001). *Generische Entwurfsmuster in der Funktionsbausteintechnik und deren Anwendung in der operativen Prozeßführung*. VDI Verlag GmbH.

[29] Epple, U. (2011). Merkmale als Grundlage der Interoperabilität technischer Systeme. *at – Automatisierungstechnik 59*, 440–450.

[30] Evertz, L. und U. Epple (2013). Laying a Basis for Service Systems in Process Control. In *ETFA 2013: IEEE 18th International Conference on Emerging Technologies and Factory Automation*, Piscataway, NJ. IEEE.

[31] George, J. (2014). Der Prolist-Workflow im eClass-Umfeld. *atp edition 1-2/2014*.

[32] Heeg, M. (2005). *Ein Beitrag zur Modellierung von Merkmalen im Umfeld der Prozessleittechnik*. VDI Verlag GmbH.

[33] Hepp, M., J. Leukel und V. Schmitz (2005). A quantitative analysis of eCl@ss, UNSPSC, eOTD, and RNTD content, coverage, and maintenance. In *ICEBE 2005: IEEE International Conference on e-Business Engineering*, S. 572–581.

[34] Höme, S., J. Grützner, T. Hadlich, C. Diedrich, D. Schnäpp, S. Arndt und E. Schnieder (2015). Semantic Industry: Herausforderungen auf dem Weg zur rechnergestützten Informationsverarbeitung der Industrie 4.0. *at Automatisierungstechnik 2*, 74 – 86.

[35] Hsu, C., G. Babin, W. Cheung, L. Rattner und L. Yee (1992). Metadatabase Modeling for Enterprise Information Integration. *Journal of Systems Integration 2*, 5–39.

[36] International Electrotechnical Commission. IEC 61360 – Common Data Dictionary. URL `http://std.iec.ch/iec61360`, (besucht am 09.12.2015). http://std.iec.ch/iec61360.

[37] International Electrotechnical Commission. IEC 61360: Standard data element types with associated classification (alle Teile).

[38] International Electrotechnical Commission. IEC 61987: Industrielle Leittechnik – Datenstrukturen und -elemente in Katalogen der Prozessleittechnik (alle Teile).

[39] International Electrotechnical Commission. IEC 62264: Integration von Unternehmensführungs- und Leitsystemen (alle Teile).

[40] International Electrotechnical Commission (2003, September). IEC TR 61131-8: Programmable controllers – Part 8: Guidelines for the application and implementation of programming languages.

[41] International Electrotechnical Commission (2011, September). IEC 61804-3 Ed. 2.0: Function blocks (FB) for process control - Part 3: Electronic Device Description Language (EDDL.

[42] International Organization for Standardization. ISO 10303: Industrielle Automatisierungssysteme und Integration - Produktdatendarstellung und -austausch (alle Teile).

[43] International Organization for Standardization. ISO 13584: Industrielle Automatisierungssysteme und Integration - Teilebibliothek (alle Teile).

[44] International Organization for Standardization, International Electrotechnical Commission (2007). ISO/IEC 13249: Information technology - Database languages - SQL multimedia and application packages (alle Teile).

[45] International Organization for Standardization, International Electrotechnical Commission (2011). ISO/IEC 9075:Information technology - Database languages - SQL (alle Teile).

[46] International Telecommunication Union (1994, Juli). ITU-T X.200: Information technology – Open Systems Interconnection – Basic Reference Model: The basic model.

[47] John, K.-H. und M. Tiegelkamp (2000). *SPS-Programmierung mit IEC 61131-3*. Springer.

[48] Kampert, D. und U. Epple (2013a). A Service Interface for Exchange of Property Information. In *IECON 2013 : 39th annual conference of the IEEE Industrial Electronics Society*, Piscataway, NJ, S. 6920–6925. IEEE.

[49] Kampert, D. und U. Epple (2013b). Dienste für den operativen Zugriff auf Merkmalinformation in der Automatisierung - Spezifikation - Integration - Anwendung. In *Automation 2013 : 14. Branchentreff der Mess- und Automatisierungstechnik*, Volume 2209 of *VDI-Berichte*, Düsseldorf, S. 61–64. VDI-Verl.

[50] Lacroix, M. und A. Pirotte (1977). Domain-Oriented Relational Languages. In *Proceedings of the Third International Conference on Very Large Data Bases, October 6-8, 1977, Tokyo, Japan.*, S. 370–378.

[51] Leitao, P., J. Mendes und A. Colombo (2008, Sept). Decision support system in a service-oriented control architecture for industrial automation. In *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, S. 1228–1235.

[52] Li, F., G. Bayrak, K. Kernschmidt und B. Vogel-Heuser (2012). Specification of the Requirements to Support Information Technology-Cycles in the Machine and Plant Manufacturing Industry. In *14th IFAC Symposium on Information Control Problems in Manufacturing (INCOM'12)*, S. 1077–1082.

[53] Manufacturing Enterprise Solutions Association (MESA) (2013). Business To Manufacturing Markup Language Release Notes Version 6.0. Technical report, Manufacturing Enterprise Solutions Association (MESA). URL `http://www.mesa.org`, (besucht am 09.12.2015).

[54] Mertens, M. (2012). *Verwaltung und Verarbeitung merkmalbasierter Informationen: vom Metamodell zur technoligischen Realisierung*. VDI Verlag GmbH.

[55] Meyer, D. (2002). *Objektverwaltungskonzept für die operative Prozessleittechnik*. VDI Verlag GmbH.

[56] Mhlanga, F., J. Wang, T. Shiau und P. Ng (1992, Jun). A query algebra for office documents. In *Systems Integration, 1992. ICSI '92., Proceedings of the Second International Conference on*, S. 458–467.

[57] NAMUR: Interessengemeinschaft Automatisierungstechnik der Prozessindustrie (2003, Februar). NA 94: MES: Funktionen und Lösungsbeispiele der Betriebsleitebene.

[58] NAMUR: Interessengemeinschaft Automatisierungstechnik der Prozessindustrie (2010, Juni). NE 100, Version 3.2: Nutzung von Merkmalleisten im PLT-Engineering-Workflow.

[59] OPC Foundation (2013). OPC Unified Architecture for ISA-95 Common Object Model Companion Specification Release 1.00. Technical report, OPC Foundation. URL http://www.opcfoundation.org, (besucht am 09.12.2015).

[60] Organization for the Advancement of Structured Information Standards (2006, Oktober). OASIS soa-rm: Reference Model for Service Oriented Architecture 1.0.

[61] Otto, B., H. Beckmann, O. Kelkar und S. Müller (2002). E-Business-Standards: Verbreitung und Akzeptanz. Technical report, Fraunhofer-Institut für Arbeitswirtschaft und Organisation. URL http://publica.fraunhofer.de/documents/N-9942.html, (besucht am 09.12.2015).

[62] PLCopen und OPC Foundation (2014). OPC-UA Client FUNCTION BLOCKS for IEC61131-3. Technical report, PLCopen und OPC Foundation. URL http://www.plcopen.org, (besucht am 09.12.2015).

[63] Prinz, J., A. Lüder, N. Suchold und R. Drath (2011). Beschreibung mechatronischer Objekte durch Merkmale. *atp edition 7-8/2011*.

[64] Robert Thurlow (2009, Mai). RFC 5531: RPC: Remote Procedure Call Protocol Specification Version 2. URL https://tools.ietf.org/html/rfc5531, (besucht am 09.12.2015).

[65] Schlütter, M., U.Epple und T. Edelmann (2009). On service-orientation as a new approach for automation environments. In *ARGESIM Report no. 35: Proceedings MATHMOD 09 Vienna - Full Papers CD Volume*.

[66] Schuppert, A. und R. Perne (2005). Data Mining mit Prozessdaten. *at - Automatisierungstechnik 53*, 342–349.

[67] Sokolov, S. und C. Diedrich (2013). Stammdaten im Engineering. *at - Automatisierungstechnik 6*, 427 – 435.

[68] Sparr, T. (1982, June). A Language for a Scientific and Engineering Database System. In *Design Automation, 1982. 19th Conference on*, S. 865–871.

[69] Sten Grüner and David Kampert and Ulrich Epple (2012, März). A Model-Based Implementation of Function Block Diagram. In *Tagungsband Modellbasierte Entwicklung eingebetteter Systeme*, München, S. 81–90. fortiss GmbH.

[70] Steusloff, H. (1994). *Prozeßleittechnik*, Chapter 8, S. 535–569. Oldenbourg Verlag.

[71] Tauchnitz, T., E. Grötsch, U. Kuhn, D. Wichmann und E. Linzenkirchner (1997). *Handbuch der Prozeßautomatisierung*, Chapter Methoden, Geräte und Systeme zur Prozeßführung, S. 15–148. Oldenbourg Verlag.

[72] Ullman, J. D. (1988). *Principles of Database and Knowledge-Base Systems*. Computer Science Press.

[73] Verein Deutscher Ingenieure (1995, Oktober). VDI/VDE 3696: Herstellerneutrale Konfigurierung von Prozeßleitsystemen – Blatt 2: Standard-Funktionsbausteine.

[74] Verein Deutscher Ingenieure (2006, August). VDI Richtlinie 5600: Manufacturing Execution Systems.

[75] Verein Deutscher Ingenieure e.V. und Zentralverband Elektrotechnik und Elektronikindustrie (2015). Referenzarchitekturmodell Industrie 4.0 (RAMI4.0). Technical report, Verein Deutscher Ingenieure e.V. und Zentralverband Elektrotechnik und Elektronikindustrie. URL http://www.zvei.org/Downloads/Automation/Statusreport-Referenzmodelle-2015-v10.pdf, (besucht am 09.12.2015).

[76] Vicknair, C., M. Macias, Z. Zhao, X. Nan, Y. Chen und D. Wilkins (2010). A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, S. 42:1–42:6.

[77] World Economic Forum (2015). Industrial Internet of Things: Unleashing the Potential of Connected Products and Services. URL http://www3.weforum.org/docs/WEFUSA_IndustrialInternet_Report2015.pdf, (besucht am 09.12.2015).

Online-Buchshop für Ingenieure

VDI nachrichten **BUCHSHOP**

Online-Shops

Fachliteratur und mehr - jetzt bequem online recherchieren & bestellen unter: www.vdi-nachrichten.com/Der-Shop-im-Ueberblick

Täglich aktualisiert: Neuerscheinungen VDI-Schriftenreihen

Im Buchshop von vdi-nachrichten.com finden Ingenieure und Techniker ein speziell auf sie zugeschnittenes, umfassendes Literaturangebot.

Mit der komfortablen Schnellsuche werden Sie in den VDI-Schriftenreihen und im Verzeichnis lieferbarer Bücher unter 1.000.000 Titeln garantiert fündig.

Im Buchshop stehen für Sie bereit:

VDI-Berichte und die Reihe **Kunststofftechnik:**

Berichte nationaler und internationaler technischer Fachtagungen der VDI-Fachgliederungen

Fortschritt-Berichte VDI:

Dissertationen, Habilitationen und Forschungsberichte aus sämtlichen ingenieurwissenschaftlichen Fachrichtungen

Newsletter „Neuerscheinungen":

Kostenfreie Infos zu aktuellen Titeln der VDI-Schriftenreihen bequem per E-Mail

Autoren-Service:

Umfassende Betreuung bei der Veröffentlichung Ihrer Arbeit in der Reihe Fortschritt-Berichte VDI

Buch- und Medien-Service:

Beschaffung aller am Markt verfügbaren Zeitschriften, Zeitungen, Fortsetzungsreihen, Handbücher, Technische Regelwerke, elektronische Medien und vieles mehr – einzeln oder im Abo und mit weltweitem Lieferservice

Die Reihen der Fortschritt-Berichte VDI:

1 Konstruktionstechnik/Maschinenelemente
2 Fertigungstechnik
3 Verfahrenstechnik
4 Bauingenieurwesen
5 Grund- und Werkstoffe/Kunststoffe
6 Energietechnik
7 Strömungstechnik
8 Mess-, Steuerungs- und Regelungstechnik
9 Elektronik/Mikro- und Nanotechnik
10 Informatik/Kommunikation
11 Schwingungstechnik
12 Verkehrstechnik/Fahrzeugtechnik
13 Fördertechnik/Logistik
14 Landtechnik/Lebensmitteltechnik
15 Umwelttechnik
16 Technik und Wirtschaft
17 Biotechnik/Medizintechnik
18 Mechanik/Bruchmechanik
19 Wärmetechnik/Kältetechnik
20 Rechnerunterstützte Verfahren (CAD, CAM, CAE CAQ, CIM ...)
21 Elektrotechnik
22 Mensch-Maschine-Systeme
23 Technische Gebäudeausrüstung

ISBN 978-3-18-5**25608**-0