KONZEPT EINES MODULBASIERTEN ENGINEERINGS IN DER ANLAGENAUTOMATISIERUNG

Vom Fachbereich 12 Maschinenwesen
der Universität Duisburg – Essen
zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
genehmigte Dissertation

vorgelegt von

Ulrich Schütten aus Essen

Juni 2003

Vorsitzender:	Prof. Dr.-Ing. habil. Volker Hans
Gutachter:	Prof. Dr.-Ing. Hans-Joachim Stracke
	Prof. Dr.-Ing. Rudolf Tracht
Tag der mündlichen Prüfung:	3. September 2003

DANKSAGUNG

Herrn Univ.-Prof. Dr.-Ing. Hans-Joachim Stracke danke ich für das Ermöglichen und die Förderungen dieser Arbeit sowie seine Unterstützung, Hilfen und Anregungen bei deren Realisierung.

Herrn Univ.-Prof. Dr.-Ing. Rudolf Tracht danke ich für die Bereitschaft das Koreferat zu übernehmen und seine Anregungen bei Durchführung der Arbeit.

Meinem Bruder Dr.-Ing. Markus Schütten danke ich für seine fachliche Unterstützung.

Mein besonderer Dank gilt meiner Frau Inga, ohne deren Hilfe eine berufsbegleitende Promotion nicht möglich gewesen wäre.

Essen, im September 2003 Ulrich Schütten

INHALTSVERZEICHNIS

1 EINLEITUNG

Mit der Entwicklung der ersten Elektromotore zum Ende des 19. Jahrhunderts hielt die Elektrotechnik Einzug in das Gebiet der Fabrikausrüstungen, das bis zu diesem Zeitpunkt eine Domäne der Mechanik und Feinwerktechnik war. Seit dieser Zeit ist die Elektrotechnik ein nicht mehr wegzudenkender Teil der Maschinen- und Anlagenausrüstung. Im Laufe der Zeit wurden der Elektrotechnik immer mehr der bis dahin mechanisch, hydraulisch oder pneumatisch gelösten Aufgaben, wie Regeleinrichtungen oder Programmsteuerwerke übertragen. Einen ungeheuren Aufschwung erlebte die Elektrotechnik seit der Entwicklung der ersten integrierten Schaltkreise und Mikroprozessoren in der zweiten Hälfte des 20. Jahrhunderts. Diese Entwicklung führte innerhalb der Elektrotechnik zur Entstehung eines neuen Fachgebietes: der Automatisierung.

Der Wert der gesamten elektrischen Ausrüstung einer Maschine konnte bereits 1995 bis zu 60 Prozent der Herstellkosten ausmachen. Nach aktuellen Untersuchungen durch den VDMA[1] erreicht allein der Softwareanteil in der Automatisierung teilweise bis zu 40 Prozent der Entwicklungsleistung im Maschinen- und Anlagenbau, wobei die Tendenz steigend ist, da sich die Kosten von der Hardware zur Software verlagern [1].

Die Anlagenbetreiber fordern von den Anlagenherstellern eine kürzere Projektdurchlaufzeit der Anlagenentwicklung, sowie niedrigere Investitionskosten und Lebenszykluskosten der Anlagen. Gleichzeitig steigen die Anforderungen an die Flexibilität des Produktes Anlage während ihres Betriebs, wie kürzere Umrüstzeiten bei einem Produktwechsel, verringerte Stillstandszeiten durch eine optimierte Stördiagnose oder die Einbindung in die Produktionslogistik durch Vernetzung.

Der Zusammenarbeit von Maschinen- und Anlagenbau und der Elektroausrüstung mit der Automatisierung kommt wegen des stetig steigenden Anteils der Automatisierung an der Gesamtwertschöpfung einer Anlage besondere Bedeutung zu. Ein Schwerpunkt der Betrachtungen dieser Arbeit liegt dabei auf der Verbesserung und Optimierung der Abläufe zwischen diesen Gebieten.

Im Zuge einer Projektabwicklung ist es erforderlich, dass zwischen den Fachgebieten Informationen über die zu errichtende Anlage ausgetauscht werden. Dabei ist zu berücksichtigen, dass nur die Informationen an die anderen Fachgebiete weitergeleitet werden, die dort auch benötigt und

[1] VDMA: **V**erband **D**eutscher **M**aschinen- und **A**nlagenbau, Frankfurt

verarbeitet werden können. Aus der Gesamtmenge, der zur Verfügung stehenden Daten, wird nur eine aufbereitete Untermenge an die Beteiligten weitergegeben.

Im Maschinen- und Anlagenbau hat sich in den letzten Jahren im Bereich der CAD-Systeme durch Einführung eines anbieterunabhängigen CAD-Formates und die Nutzung von PDM-Systemen eine einheitliche Softwarelandschaft ausgebildet, die es ermöglicht, unternehmensweit auf die Informationen, Daten und Zeichnungen zuzugreifen [2].

In der Elektrotechnik und Automatisierung haben sich für die Projektierung, Programmierung und Zeichnungserstellung unterschiedlichste Softwaresysteme als Spezialwerkzeuge zur Lösung eingegrenzter Aufgaben etabliert. Das Merkmal der Software ist, dass damit einzelne Arbeiten und Vorgänge der Entwicklung unterstützt werden, was für sich gesehen zu einer Erhöhung der Produktivität führt. Auf Grund ihrer Entstehungsgeschichte bilden die Systeme Insellösungen im Gesamtplanungsprozess der Automatisierung.

Von den Anwendern wird gefordert, dass die Systeme im gesamten integrierten Entwicklungsprozess aufeinander abgestimmt zusammenarbeiten. Die zurzeit auf dem Markt angebotenen Softwaresysteme zur Projektierung in der Automatisierung wie Elektro-CAD (E-CAD), SPS-Programmierung und Leitsystementwicklung erfüllen die Voraussetzungen für einen Datenaustausch untereinander nur unvollständig. Ein übergreifender Datenaustausch mit Fremdsystemen, wie er im Anlagenbau benötigt wird, ist mit den Systemen nicht möglich [3].

Im Folgenden wird eine Datenstruktur, basierend auf einem offenen Standard für den Informationsaustausch, definiert. Die entwickelte Datenstruktur ermöglicht nicht nur den reinen Informationsaustausch, sondern lässt sich automatisch aus den Systemen zur Datenverarbeitung und Projektierung heraus erzeugen und dient nach einem Import der automatischen Weiterverarbeitung.

Auf der Basis des entwickelten Austauschformates wird das Konzept einer automatisierten Weiterverarbeitung der Informationen auf dem Gebiet der Automatisierung erarbeitet. Ziel des Konzeptes ist es, aus den ausgetauschten Daten das Basisengineering für

- Stromlaufpläne,
- SPS-Programme und
- Leitsystemfunktionen

generisch mit dem in dieser Arbeit entwickelten Software-Werkzeug **dbEngineer**[1] zu erzeugen. Mit dem **dbEngineer** wird datenbankbasiert der Engineeringprozess in der Automatisierung unterstützt.

Die Herleitung des Konzeptes beginnt zunächst mit der Problemstellung. Dazu werden der Stand der Technik und der typische Ablauf der Anlagenplanung an Hand eines Strukturplans analysiert. Sowohl die Aufgaben der Automatisierung im Anlagenbau sowie die dazu erforderlichen Planungswerkzeuge zur Projektierung und die daraus entstehende typische Vorgehensweise des Projektierungsprozesses werden vorgestellt. Die Schwachpunkte der auf dem Markt befindlichen Planungswerkzeuge werden aufgezeigt, womit sich die Anforderungen an das Software-Werkzeug **dbEngineer** ableiten lassen. Die Darstellung der momentan verfügbaren Software- und Datenstrukturen wird die Möglichkeiten zur Gestaltung der Software-Architektur des **dbEngineers** erläutern.

Nach Erarbeitung eines allgemeinen Verständnisses für die Belange der Projektierung wird zunächst der Grundgedanke des automatischen Generierens von Planungsunterlagen auf der Basis einer Datenaustauschdatei und Modulbibliothek eingeführt. Auf dieser Basis wird das Konzept des **dbEngineers** mit seinen Software- und Datenstrukturen entwickelt und vorgestellt. Es wird dargestellt, wie mit dem entwickelten Konzept die Probleme des typischen Projektierungsablaufs überwunden werden können.

Im Rahmen einer Dissertation ist eine vollständige softwaremäßige Umsetzung des vorgestellten **dbEngineers** nicht möglich. Der Aufwand für die vollständige Softwareerstellung des **dbEngineers** beträgt mehrere Mannjahre. Um Funktionsfähigkeit des Konzepts zu demonstrieren, zeigt die abschließende Anwendung des Konzepts **dbEngineer** die programmtechnische Realisierung eines Prototypen zur Kopplung an das SPS-Programmiersystem SIMATIC STEP 7.

[1] **dbEngineer**: **d**aten**b**ankbasiertes **Engineer**ing

2 ISTZUSTAND DER ANLAGENPLANUNG

Im Großanlagenbau, wie der Chemischen Industrie, der Hütten- und Walzwerkstechnik oder dem Kläranlagenbau, sind mehrere Parteien beteiligt: der Investor, der oder die Kontraktoren und die Lieferanten. Je nach Erfordernis beauftragt der Investor einen Gesamtverantwortlichen als Generalunternehmer mit der Errichtung der gewünschten Anlage oder er behält sich die Koordination selbst vor und erteilt den Kontraktoren für die jeweiligen Teilaufgaben eigene Aufträge. Es ist unerheblich welches Vergabeverfahren gewählt wird, weil in der modernen arbeitsteiligen Welt die einzelnen Teilaufgaben durch verschiedene Lieferanten bearbeitet werden. Wegen der Verteilung eines Gesamtprojekts auf verschiedene Spezialisten ist eine einheitliche EDV-Landschaft und Datenhaltung nicht zu erwarten [3].

In den folgenden Abschnitten werden der Projektablauf einer Anlagenplanung, der Datenaustausch in Form und Inhalt, die Struktur- und Referenzkennzeichnung einer Anlage, sowie die Gebiete der Automatisierung mit ihren Planungswerkzeugen vorgestellt. Es schließt sich eine Betrachtung der Probleme, die sich im Zuge der Anlagenplanung ergeben, an. Später wird ein Konzept zur Überwindung dieser Schwierigkeiten vorgestellt.

2.1 Projektablauf

Für die Betrachtung des Istzustandes im Maschinen- und Anlagenbau ist es wichtig den Ablauf und das Zusammenwirken der Beteiligten bei der Projektabwicklung zu kennen. Die Idee zum Bau einer neuen oder zum Umbau einer existierenden Anlage entsteht beim Investor als Betreiber, der mit der geplanten Investition auf die Erfordernisse seines Marktes reagiert. Dabei kann es sich um die Fertigung neuer Produkte, höherer Qualität, kürzerer Fertigungszeiten oder niedrigerer Stückkosten handeln.

Bei den daraus entstehenden Großanlagen handelt es sich durchweg um komplexe Systeme, die einen geringen Wiederholgrad von Teilanlagen besitzen. Die Art der Abwicklung einzelner Projekte unterscheidet sich deshalb stark voneinander. Trotzdem gilt für jedes Projekt ein gemeinsamer Grundrahmen für die Vorgehensweise der Projektabwicklung. Die typische Projektabwicklung im

Großanlagenbau wurde von der NAMUR[1] in ihrem Arbeitsblatt 35 dargelegt und zur allgemeinen Verwendung definiert.

Das Arbeitsblatt 35 teilt den Ablauf eines Projektes mit Hilfe eines Standard-Projektstrukturplanes in Funktionsbereiche und deren zeitlicher Abfolge ein. Mit Hilfe dieser Gliederung und den im Arbeitsblatt angegebenen Erläuterungen kann der Strukturplan nach NAMUR allgemein für alle Projekte im Anlagenbau genutzt werden.

Die einzelnen Stufen einer Projektabwicklung von der Studie bis zur fertigen und abgeschlossenen Anlage zeigt Abbildung 2-1. Die Darstellung entspricht dem NAMUR-Arbeitsblatt 35, das Hilfestellung für die Projektabwicklung bietet und den gesamten Planungsprozess in sieben verschiedene typische Planungsphasen mit den entsprechenden Projektaktivitäten einteilt.

Standard-Projektstrukturplan						
1 Studie	**2 Vorplanung**	**3 Basisplanung**	**4 Ausführungsplanung**	**5 Anlagenerrichtung**	**6 IBS**	**7 Projektabschluss**
1.1.Aufgabenstellung analysieren	2.1 Konzept erarbeiten	3.1 Basisanforderungen spezifizieren	4.1 Ausführung spezifizieren	5.1 Lieferungen und Leistungen bestellen	6.1 Betriebsbetreuung einarbeiten	7.1 Gewerke auswerten
1.2 Kosten kalkulieren	2.2 Konzessionsunterlagen erstellen	3.2 Termin- und Kostenplanung durchführen	4.2. Detailplanung durchführen	5.2 Komponenten errichten und prüfen	6.2 Übergabe von der Planung an den Betrieb durchführen	7.2 Projekt nachkalkulieren
			4.3 Ausführung dokumentieren	5.3 Anlage montieren	6.3 Anlage inbetriebsetzen	7.3 Projekt formal abschliessen
					6.4 Revision der Dokumentation durchführen	
Ziel						
Die durchführbare Anlage	**Die genehmigungsfähige Anlage**	**Die ausschreibbare Anlage**	**Die errichtbare Anlage**	**Die funktionsfähige Anlage**	**Die produktionsfähige Anlage**	**Das bewertete und abgerechnete Projekt**

Abbildung 2-1: Projektstrukturplan nach NAMUR Arbeitsblatt 35

[1] NAMUR: Interessengemeinschaft Prozessleittechnik der chemischen und pharmazeutischen Industrie, Leverkusen; gegründet 1949 als **N**ormen**a**rbeitsgemeinschaft für **M**ess- **u**nd **R**egeltechnik in der chemischen Industrie

Bei der Betrachtung des Strukturplans ist es wichtig zu erkennen, dass die Vorgänge nicht nur strukturiert, sondern auch serialisiert werden müssen. Dies bedeutet, dass eine Stufe abgeschlossen sein muss, bevor die Tätigkeiten der nächsten oder übernächsten Stufe begonnen werden können.

Der Investor führt zunächst für sich die Stufen 1 und 2 des Strukturplanes durch und fragt mit den bis dahin erstellten Unterlagen Anlagenbauer an. Der beauftragte Anlagenbauer beginnt für sich, basierend auf einem eigenen Standardprojektplan, mit seiner Projektbearbeitung. In seiner Phase 3 hat er die Unterlagen soweit erstellt, dass entweder der Anlagenbauer oder der Investor die Automatisierung mit den bis zu diesem Zeitpunkt erstellten Unterlagen anfragen und beauftragen kann. Auf einer Zeitachse qualitativ aufgetragen, ergeben sich dabei folgende Abhängigkeiten, wie in Abbildung 2-2 dargestellt.

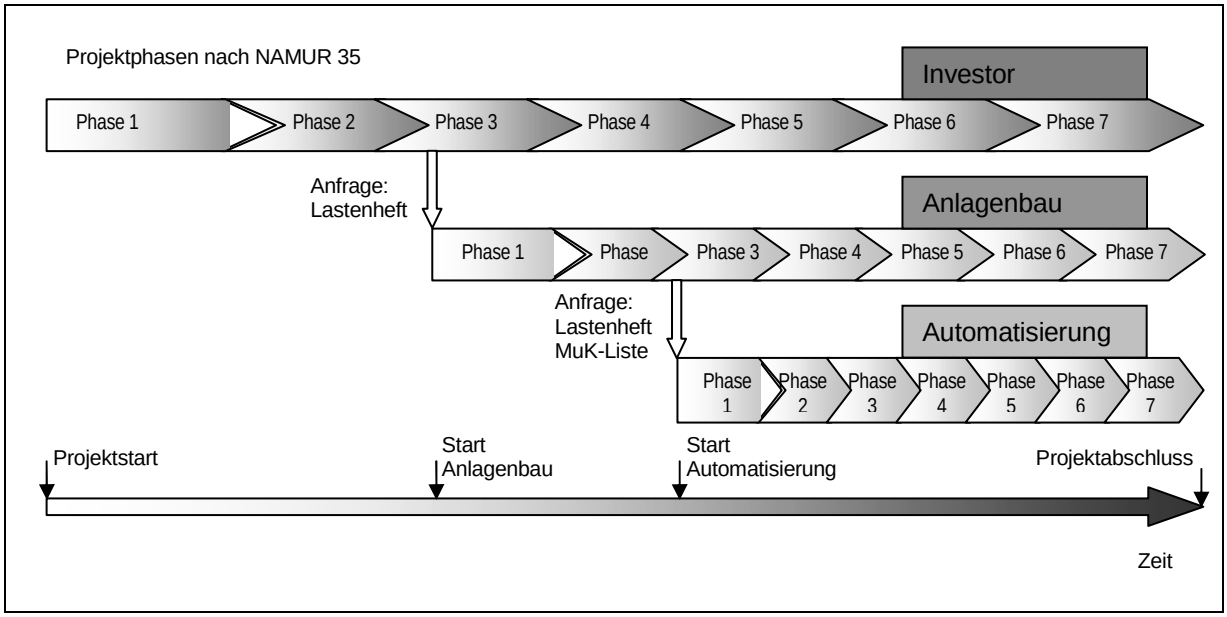

Abbildung 2-2: Startzeitpunkte für verschiedene Gewerke im Gesamtablauf

Die Automatisierung kann erst mit ihrer Projektarbeit beginnen, wenn die anderen Projektbeteiligten ihre Arbeit soweit vorangetrieben haben, dass die Spezifikationen für die Automatisierung vorliegen: dazu zählen unter anderem Anzahl und Leistungsbedarf von Antrieben, Aktorik, Sensorik sowie Basisfunktionspläne und die Einbindung in die übergeordnete Produktionslogistik. Der Automatisierung verbleibt zur Abwicklung ihrer Entwicklungsleistungen, die mittlerweile fast denselben Umfang wie im Anlagenbau angenommen haben, deutlich weniger Zeit als dem Anlagenbau.

Der kritische Pfad eines typischen Projektterminplans nach Pert führt über die Automatisierung, das heißt, jede Verzögerung innerhalb der Automatisierung führt zu einer Verzögerung des Gesamtprojekts. Dieser Entwicklung kann durch einen erhöhten Personaleinsatz entgegengewirkt werden. Der VDMA erwartet für das Jahr 2010 im deutschen Maschinenbau einen zusätzlichen Bedarf gegenüber dem Jahr 2000 bei Elektroingenieuren von 38,2% und bei Informatikern von 38,9%, während der Zusatzbedarf an Maschinenbauingenieuren nur um 19,2% steigen wird [4].

Durch einen höheren Personaleinsatz kann, wegen des dadurch entstehenden zusätzlichen Abstimmungsbedarfs im Team, das Zeitproblem nicht allein gelöst werden. Deshalb wird in einem späteren Kapitel die Planung in der Automatisierung mit deren Problemfeldern gesondert betrachtet werden.

Zunächst sollen die Dokumente des Datenaustausches vorgestellt werden. In Abbildung 2-2 werden nicht nur die zeitlichen Abhängigkeiten gezeigt, sondern auch die für die Anlagenplanung zentralen Dokumente des Informationsaustausches: Lastenheft und Motoren- und Komponentenliste. Wegen ihrer Bedeutung werden die Dokumente in einem eigenen Kapitel beschrieben.

2.2 Datenaustausch im Anlagenbau

Für den Informationsfluss bedeutet die zeitliche Abfolge, wie im NAMUR Standard-Projektstrukturplan dargestellt, dass der jeweils Nachfolgende auf der Zeitachse von seinem Vorgänger Informationen und Unterlagen erhalten muss, damit er seine Planung beginnen kann. Der typische Informationsaustausch erfolgt über ein Lastenheft nach VDI[1]/VDE[2] Richtlinie 3694 [5] und über Austauschlisten mit den technischen Spezifikationen und dem Mengengerüst der Anlage.

Als industrieweit anerkannter Standard für eine Austauschliste gilt dabei die Motoren- und Komponentenliste (MuK-Liste) [6]. Die MuK-Liste wurde in den 70er-Jahren des letzten Jahrhunderts von deutschen Maschinen- und Anlagenbauern, Elektroausrüstern sowie Investoren entwickelt, um eine gemeinsame Plattform für die technische Ausarbeitung in der Investitionsgüterindustrie zu definieren. Sie enthält Angaben über die vorgesehenen Antriebs- und Motordaten,

[1] VDI: **V**erband **d**eutscher **I**ngenieure e.V., Düsseldorf

[2] VDE: **V**erband **d**er **E**lektrotechnik Elektronik Informationstechnik e.V. , Frankfurt

Daten über Betriebsmittel der Prozess- und Bedienperipherie, Angaben über Speisung, Hauptstromkreise sowie Steuerung und Regelung.

Die MuK-Liste folgt dem Gedanken, eine Anlage in Einzelbewegungen funktional zu zerlegen. Die Maschinenbewegungen werden in einer Anlage von aktiven Komponenten wie Motore oder Ventile, der Aktorik, ausgeführt. Die MuK-Liste weist eine Bewegungsfunktion genau einem Aktor zu. In einer Anlage wird der Zustand einer Bewegung durch die zugeordnete Sensorik, wie Initiatoren, Weggeber oder Messaufnehmer physikalischer Größen (Druck, Durchfluss, Temperatur usw.) überwacht und zurückgemeldet. Die MuK-Liste fasst den Aktor mit seiner Sensorik zu einer Funktionsgruppe, die auch als elektro-mechanisches Grundsystem bezeichnet wird, zusammen. Das elektro-mechanische Grundsystem wird als eine Einheit dargestellt und beschrieben.

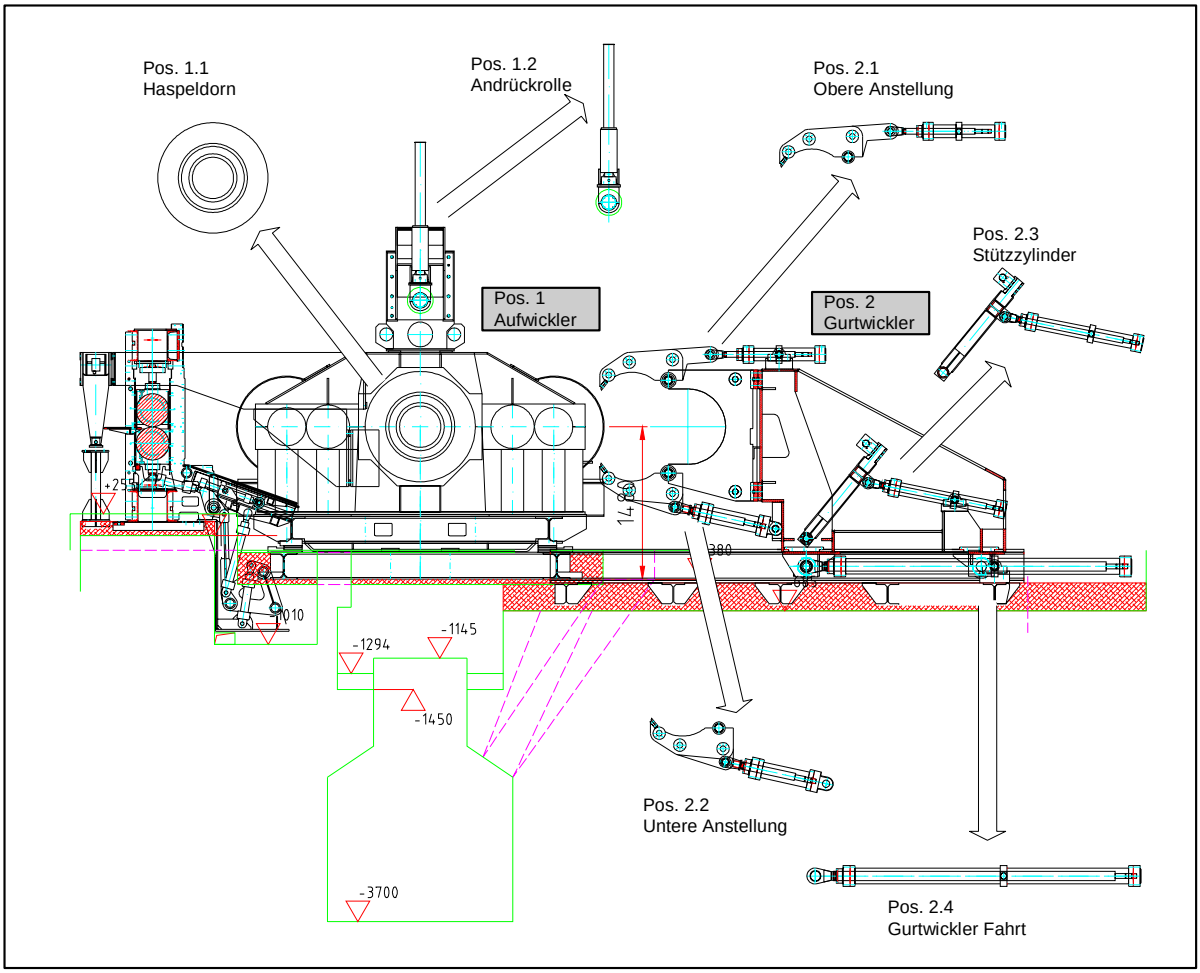

Abbildung 2-3: Funktionstechnische Dekomposition einer Anlage für die MuK-Liste

Abbildung 2-3 zeigt in einem Ausschnitt für eine Walzwerkanlage die Aufteilung in Funktionsgruppen. Dargestellt ist der Ausschnitt *Aufwickelbereich* einer Bandanlage mit den Funktionsbereichen *Aufwickler* und *Gurtwickler*. Der Teilbereich *Aufwickler* besteht dabei, wie verdeutlicht, aus zwei Einzelfunktionen:

- dem Haspeldornantrieb und
- der Andrückrolle.

Der Teilbereich *Gurtwickler* beinhaltet vier Einzelfunktionen:

- die oberen Anstellung,
- die unteren Anstellung,
- den Stützzylinder und
- die Gurtwickler Fahrt.

Der Aufbau der MuK-Liste gruppiert die Funktionsbereiche ausgehend von der Gesamtanlage, über Teilbereiche bis zu den Einzelfunktionen. Bei der Gliederung durch die MuK-Liste erfährt der konstruktive Aufbau einer Anlage keine Berücksichtigung.

Position	**Antriebs - und Motordaten**							**Speisung**	**Hauptstromkreis**		**Steuerung Regelung**		**Zubehör Anbauten**		**Bemerkung**	**Ind. Rev.**
Mechan. Ausrüstung Elektr. Ausrüstung	Anzahl Liefer-schlüssel	Art	Leistung kW kVA	Drehzahl /min	Nennbetriebsart Schaltspiele/Std.	Bauform Schutzart IP Kühlung	Fabrikat Typ	Art Volt Hz	Anzahl	Art Volt	Art	Ort der Betätigung	Anzahl	Art		Mech. Elektr.
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1. Aufwickler																
1.1 Haspeldorn	1	MKL	250	457/1800	S1	IM B3 IP54 IC01		N-DS 500 50	1	GFS	S-EL		3 1	FT BN	PT 100	
1.2 Andrückrolle	1	YVH2	0,018		S1			N-GS 24 DC	1	KR	S-EL		1 1	SBE SBE	Abgestellt Angestellt	
2. Gurtwickler																
2.1 Obere Anstellung	1	YVH2	0,018		S1			N-GS 24 DC	1	KR	S-EL		1 1	SBE SBE	Abgestellt Angestellt	
2.2 Untere Anstellung	1	YVH2	0,018		S1			N-GS 24 DC	1	KR	S-EL		1 1	SBE SBE	Abgestellt Angestellt	
2.3 Stützzylinder	1	YVH2	0,018		S1			N-GS 24 DC	1	KR	S-EL		1 1	SBE SBE	Abgestellt Angestellt	
2.4 Gurtwickler Fahrt	1	YVH2	0,018		S1			N-GS 24 DC	1	KR	S-EL		1 1	SBE SBE	Angefahren Zurückgefahren	
Motoren- und Komponentenliste Hauptantriebe	**Kennwort:**					**Maschinenbaufirma:**							**Nr.:**		**Ind./Rev.:**	**Blatt: 1**
	Anlage: Durchlaufofen Ertüchtigung E-Ausrüstung					**Elektrofirma:**							**Nr.:**		**Ind./Rev.:** **Ind./Rev.:**	**Blätter: 1**

Abbildung 2-4: Ausschnitt einer typischen MuK-Liste für die Beispielanlage (Abbildung 2-3)

Abbildung 2-4 zeigt einen Ausschnitt aus einer typischen MuK-Liste, wie sie allgemein verwendet wird. Der Aufbau der Liste unterscheidet sich wegen der fehlenden Normung von Firma zu Firma in einigen Details. Die Spalten der MuK-Liste, Abkürzungen und deren Bedeutung werden gemäß Tabelle 2-1 benutzt. Die angeführte MuK-Liste bezieht sich auf das Beispiel von *Aufwickler* und *Gutwickler* und beschreibt dessen Ausrüstung.

Spalte	Inhalt	Einheit	Bedeutung	Beispiel
1-17	**Zeile für die Benennung**			
1	**Position**			
oben	Mech. Ausrüstung		Kennzeichen	
unten	Elektr. Ausrüstung		Kennzeichen	
2-8	**Antriebs- und Motordaten**			
2	Anzahl/Lieferschlüssel		Zusatz	kein=Elektrik, X=Mechanik
3	Art		Aktor	MKL= Kurzschlussläufermotor YVH2= Hydraulikventil 2-Spulig
4	Leistung	kW		elektrische Leistungsaufnahme
5	Drehzahl	min^{-1}		
6 oben	Nennbetriebsart			nach VDE 530 Teil 1
unten	Schaltspiele/h	c/h		
7 oben	Bauform IM..			Bauform von Motoren
mitte	Schutzart IP..			nach IEC
unten	Kühlung IC			Kühlart einer Maschine nach IEC
8	Fabrikat Typ			
9	**Speisung**			
	Art/Spannung/Frequenz	/V/Hz	für Art:	N-DS = Netz Drehstrom N-GS = Netz Gleichstrom
10-11	**Hauptstromkreis**			
10	Anzahl			
11	Art/Spannung	/V	für Art:	KA = Antrieb mit einer Richtung KR = Antrieb mit zwei Richtungen GFS = Frequenzumformer
12-13	**Steuerung, Regelung**			
12	Art			S-EL = Einzelsteuerung S-FS = Sequenz-/Folgesteuerung
13	Ort der Betätigung		Steuerstelle	Referenzkennzeichen
14-15	**Zubehör, Anbauten**			
14	Anzahl			
15	Art			BN = Drehzahlgeber FT = Temperaturwächter SBE = Näherungsschalter
16	**Bemerkungen**			
17	**Index**			
oben	mechanisch			
unten	elektrisch			

Tabelle 2-1: Bedeutung von Spalten und Abkürzungen in der MuK-Liste [6]

In der Tabelle sind in der Spalte *Beispiel* nur die für MuK-Liste nach Abbildung 2-4 benutzen Begriffe angegeben, für die vollständigen Angaben zu Benutzungshinweisen, Erläuterungen und Abkürzungen sei auf die *Anleitung zur Motoren und Komponentenliste* [6] verwiesen.

Ursprünglich wurde die MuK-Liste zur Ausschreibung und Anfragebearbeitung von Elektroausrüstungen konzipiert. Die MuK-Liste wird vom Investor oder dem Anlagenbauer im Rahmen der Basisplanung erstellt und dem Lieferanten der Elektroausrüstung übergeben. Es wurde aber schnell der Nutzen einer gemeinsamen Informationsplattform zwischen Maschinenbau und Elektroausrüstung erkannt, so dass heute viele Investoren die MuK-Liste als verbindliche technische Unterlage in ihren Werkvorschriften aufführen. Sie wird deshalb, über die Anfrage und Ausschreibung hinaus, während der gesamten Errichtungsphase von den Lieferanten bearbeitet und gepflegt [7]. Dadurch wird die MuK-Liste zu einem zentralen Dokument des Informationsaustausches im Anlagenbau, wie Abbildung 2-5 verdeutlicht.

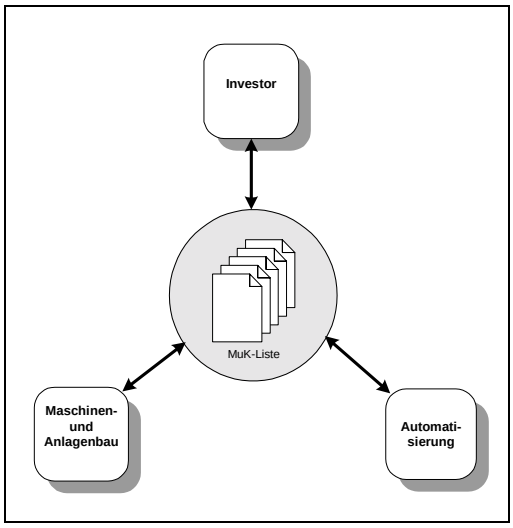

Abbildung 2-5: MuK-Liste als zentrales Dokument im Informationsaustausch

Als Datenformate für Austauschlisten haben sich in den letzten Jahren die Formate der Microsoft Officewelt etabliert, wobei insbesondere das Tabellenformat von Microsoft Excel Verwendung findet [8].

2.3 Anlagenaspekte nach DIN EN 61346

Die Zusammenarbeit zwischen Anlagenbau und Automatisierung wird durch eine unterschiedliche Sichtweise auf die zu errichtende Anlage erschwert. Der Anlagenbau orientiert sich bei der Strukturierung der Anlage in Teilanlagen, Baugruppen und Einzelteilen an den konstruktiven und fertigungstechnischen Grundsätzen. Demgegenüber strukturiert die Automatisierung eine Anlage nach funktionstechnischen Gesichtspunkten: welche Teile arbeiten für eine Funktion zusammen. Die

Unterschiede traten bereits bei der Vorstellung der MuK-Liste als zentralem Dokument der Anlagenplanung auf. DIN EN 61346 [9] trägt dieser unterschiedlichen Sichtweise mit der Einführung von *Aspekten* Rechnung. DIN EN 61346 definiert drei Aspekte für ein Objekt (Komponente oder Bauteil):

- *Produktsaspekt*: Strukturiert die Anlage nach konstruktiven Gesichtspunkten. Entspricht der Sichtweise des Anlagenbaus.
- *Funktionsaspekt*: Strukturiert die Anlage nach funktionstechnischen Gesichtspunkten. Entspricht der Sichtweise der Automatisierung.
- *Ortsaspekt*: Strukturiert die Anlage in Ortsbereiche; beispielsweise in ein dreidimensionales Koordinatensystem zum Auffinden von Objekten in einer komplexen Großanlage. Entspricht der Sichtweise des Investors.

Mit Hilfe der DIN EN 61346 lassen sich die einzelnen Aspekte als Strukturbäume gliedern. Jeder Aspekt eines Objekts kann durch denselben Aspekt anderer Objekte beschrieben werden. Diese anderen Objekte werden zu Teilobjekten des beschriebenen Objekts. Das Ergebnis der wiederholten Unterteilung desselben Aspektes des identifizierten Objekts kann als Baum dargestellt werden, wie in Abbildung 2-6 gezeigt [9]. Durch die Anwendung dieser Regel wird, ausgehend von der Gesamtanlage, das Bauteil in der untersten Ebene erreicht, das sich seinerseits nicht weiter unterteilen lässt.

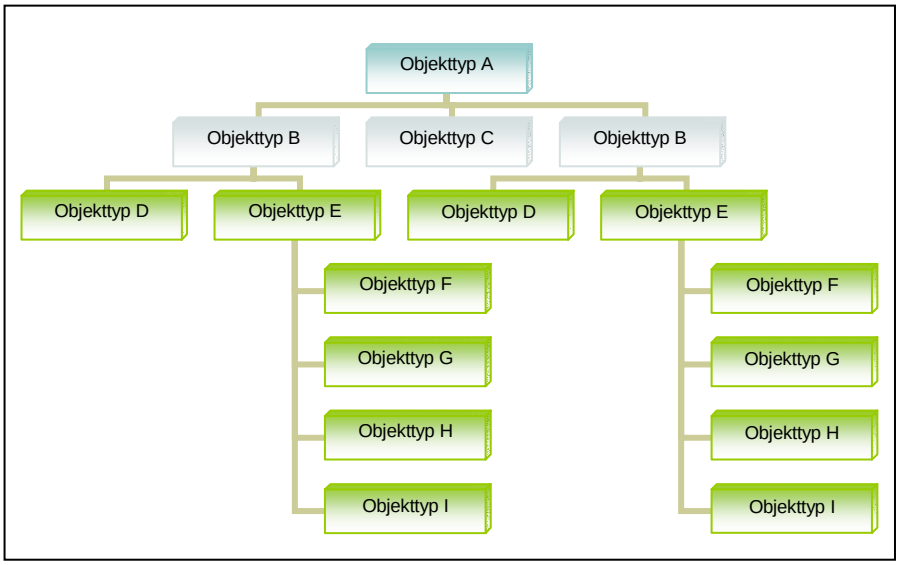

Abbildung 2-6: Strukturbaum in einem Aspekt des Objekttyps A [9]

Damit die Bauteile in den einzelnen Aspekten identifiziert werden können, vergibt die DIN ein eindeutiges Referenzkennzeichen. Gemäß DIN EN 61346 muss ein Referenzkennzeichen, welches einem Objekt zugeordnet ist, aus einem Vorzeichen bestehen, gefolgt von wahlweise

- einem Buchstabencode
- einem Buchstabencode, gefolgt von einer Zahl oder
- einer Zahl.

Für die Aspekttypen müssen die Vorzeichen zur Identifikation der Referenzkennzeichen folgende sein:

- „=“, wenn es sich auf Funktionsaspekte eines Objekts bezieht;
- „-“ , wenn es sich auf Produktaspekte eines Objekts bezieht;
- „+“, wenn es sich auf Ortsaspekte eines Objekts bezieht.

Abbildung 2-7 zeigt Beispiele von Referenzkennzeichen und deren Syntax. [9].

Funktionsbezogenes Referenzkennzeichen	Produktbezogenes Referenzkennzeichen	Ortsbezogenes Referenzkennzeichen
= A1	- A1	+ G1
= ABC	- RELAY	+ RM
= 122	- 561	+ 101

Abbildung 2-7: Beispiele für Referenzkennzeichen [9]

Die aufgeführten Unterschiede der Betrachtungsweise bei den verschiedenen Aspekten und den Fachgebieten sollen am Beispiel der Abbildung 2-8 erklärt werden. Die schematische Darstellung zeigt zwei Anlagenteile A und B zwischen denen ein elektrisch angetriebenes Fahrzeug Teile transportiert. Am Ende der Fahrzeugfahrbahn befinden sich im Bereich der Anlage A und B jeweils Wegendschalter (Initiatoren[1]), die die Fahrt abschalten und das Fahrzeug stoppen.

[1] Initiator: Berührungslos arbeitender elektronischer Endlagenschalter, Näherungsschalter

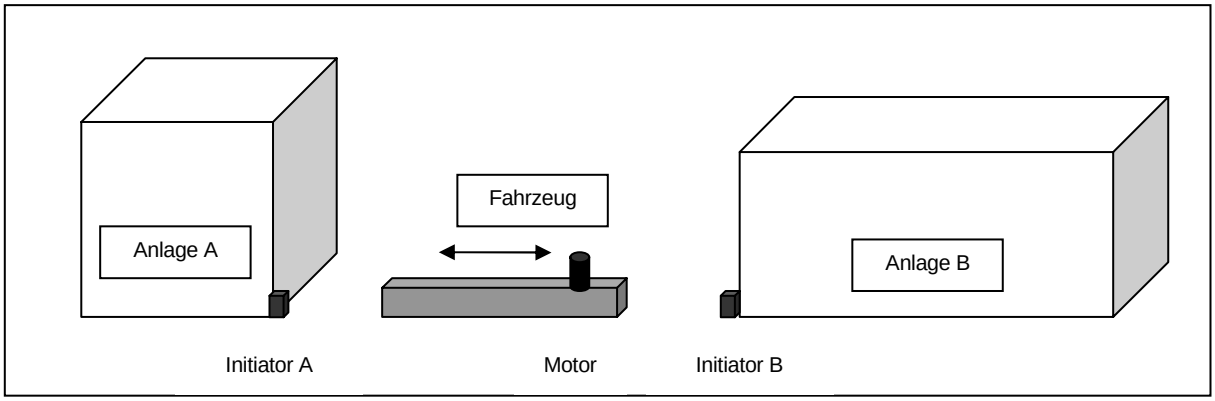

Abbildung 2-8: Schematische Darstellung einer Anlage bestehend aus Anlage A und B sowie einem Fahrzeug

Die technischen Daten für den Fahrantrieb des Transportfahrzeuges in eine MuK-Liste eingetragen zeigt Abbildung 2-9. In der MuK-Liste werden die Bauteile einer Funktion oder Funktionsgruppe zusammengefasst aufgeführt [6].

Position	**Antriebs - und Motordaten**								**Speisung**	**Hauptstromkreis**		**Steuerung Regelung**		**Zubehör Anbauten**		**Bemerkung**	**Ind. Rev.**
Mechan. Ausrüstung	Anzahl	Art	Leistung kW	Drehzahl	Nennbetriebsart	Bauform Schutzart	Fabrikat	Art	Anzahl	Art	Art	Ort der	Anzahl	Art		Mech.
Elektr. Ausrüstung	Lieferschlüssel		kVA	/min	Schaltspiele/Std.	IP Kühlung	Typ	Volt Hz		Volt		Betätigung				Elektr.
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1. Fahrzeug																
1.1 Fahrantrieb	1	MKL	15	1500	S9	IM B5 IP54 IC01		N-DS 400 50	1	KR	S-EL		1 1	SBE SBE	Endlage Anlage A (links) Endlage Anlage B (rechts	
Motoren- und Komponentenliste Hauptantriebe		**Kennwort:**					**Maschinenbaufirma:**						**Nr.:**		**Ind./Rev.:**	**Blatt: 1**
		Anlage: Muster für Dissertation E-Ausrüstung					**Elektrofirma:**						**Nr.:**		**Ind./Rev.: Ind./Rev.:**	**Blätter: 1**

Abbildung 2-9: MuK-Liste für das Fahrzeug

Wie die Strukturbäume der einzelnen Aspekte für das Beispiel aufgebaut sind, wird in den folgenden Kapiteln erläutert.

2.3.1 Der Produktaspekt

Die Anlagenkonstruktion gliedert für sich eine Anlage nach Gesichtspunkten, wie die Anlagenbereiche fertigungs- und konstruktionstechnisch zusammenhängen. Die CAD-Systeme des Anlagenbaus stellen dafür eine Teilehierarchie zur Verfügung. In der Teilehierarchie wird angegeben, aus welchen Baugruppen und Subbaugruppen sich ein Anlagenteil zusammensetzt. Diese Form der Strukturierung entspricht dem Produktaspekt der DIN.

In der Anlagenkonstruktion ergibt sich für das angeführte Beispiel eine Anlagenhierarchie[1] in Baumstruktur gemäß Abbildung 2-10. Die Wegendschalter sind den Anlagenbereichen zugeordnet, wo sie auch eingebaut werden, während der Motor Teil des Fahrzeuges ist. Die Referenzkennzeichnung der Bauteile wird über den Strukturbaum gebildet, und dient der eindeutigen Identifikation der Bauteile. Das so gebildete Kennzeichen wird beispielsweise in Stücklisten und Funktionsbeschreibungen benutzt. Üblicherweise erhält ein Bauteil in der errichteten Anlage ein graviertes Kennzeichnungsschild mit dem Referenzkennzeichen, um dem Servicepersonal während des Lebenszyklus einer Anlage die eindeutige Bestimmung des Bauteils zu ermöglichen [13].

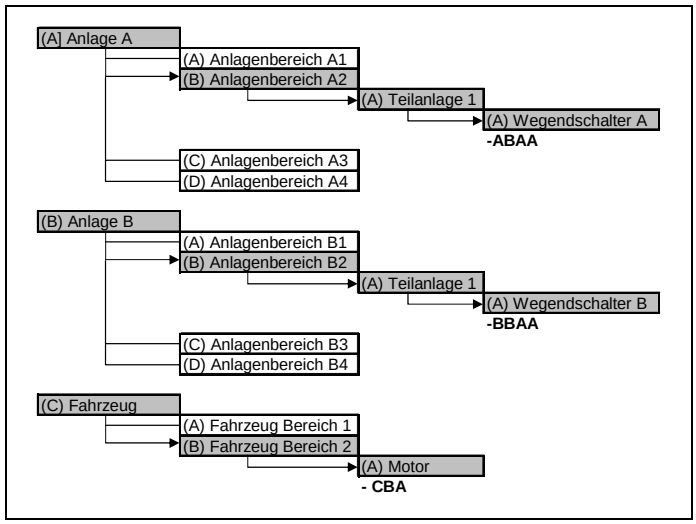

Abbildung 2-10: Anlagenstruktur im Anlagenbau (Produktaspekt) mit Referenzkennzeichen

Im Rahmen der Projektabwicklung und des zeitlichen Ablaufs der Gewerke (siehe Kapitel 2.1 *Projektablauf*) erstellt der Anlagenkonstrukteur, basierend auf seinem Konstruktionsstand (Basisplanung), die MuK-Liste mit der die Elektrische Ausrüstung der Anlage angefragt wird. Die Anlagenkonstruktion benötigt ab diesem Zeitpunkt zwei unterschiedliche Sichtweisen auf die Anlage: den Produktaspekt und den Funktionsaspekt.

[1] Die Struktur der Hierarchie ist hier zur Illustration willkürlich gewählt, Tiefe und Aufbau der Hierarchie werden bei realen Anlagen abweichen.

2.3.2 Der Funktionsaspekt

Bei der funktionalen Gliederung in der Automatisierung ergibt sich für das Beispiel des Fahrzeugs eine Struktur gemäß Abbildung 2-11[1]. Die Aufteilung orientiert sich dabei nicht an konstruktiven oder örtlichen Zusammenhängen, sondern ausschließlich am funktionalen Zusammenspiel der Teile. Die Gliederung einer Anlage nach funktionalen Gesichtspunkten ist für die Elektrotechnik durch die DIN EN 61346-1 vorgegeben. Die Automatisierung setzt voraus, dass eine Anlage in Einzelfunktionen zerlegt wird, die ihrerseits nicht mehr weiter atomisiert werden können. Bei der Dekomposition der Anlage entsteht eine Anlagenhierarchie in Baumstruktur, die alle Teilfunktionen beinhaltet. In der DIN EN 61346-1 ist ein Bezeichnungssystem der *Funktionsaspekt* (Anlagenkennzeichen) für eine derartige Baumstruktur angegeben, sie findet für die Strukturierung von Stromlaufplänen ihre Anwendung.

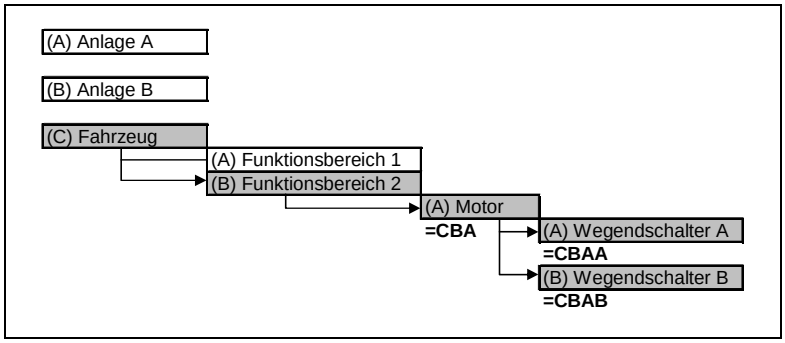

Abbildung 2-11: Anlagenstruktur als Funktionsaspekt mit Referenzkennzeichen

In Abbildung 2-11 bildet der Motor mit seinen Wegendschaltern eine eigene Funktionsgruppe, die dem Fahrzeug zugeordnet ist. Wie beim *Produktaspekt* wird dem Bauteil für den *Funktionsaspekt* ein Referenzkennzeichen zugeordnet, das zur Identifikation in der Automatisierung dient. In der Anlage wird das Bauteil auch mit dieser Kennzeichnung beschriftet.

Der Vergleich der drei Strukturen von *Produktaspekt, Funktionsaspekt* und MuK-Liste zeigt, dass die Gliederung der MuK-Liste am *Funktionsaspekt* angelehnt ist, ohne jedoch dessen Stringenz aufzugreifen.

[1] Die Struktur der Hierarchie ist hier zur Illustration willkürlich gewählt, Tiefe und Aufbau der Hierarchie werden bei realen Anlagen abweichen.

2.3.3 Der Ortsaspekt

Eine ortsbezogene Struktur basiert auf der topographischen Anordnung eines Systems und/oder der Umgebung, in der sich ein System befindet. Eine ortsbezogene Struktur zeigt die Unterteilung in Bestandteil-Objekte im Hinblick auf den *Ortsaspekt*, ohne notwendigerweise Produkte und/oder Funktionen in Betracht zu ziehen. Ein Ort kann eine beliebige Anzahl von Produkten enthalten. Eine ortsbezogene Struktur darf Orte wiederholt unterteilen. Das Referenzkennzeichen des *Ortsaspekts* beschreibt, wo sich ein Objekt physikalisch befindet [9].

Für das Beispiel nach Abbildung 2-8: *Schematische Darstellung einer Anlage bestehend aus Anlage A und B sowie einem Fahrzeug* ergibt sich eine Strukturierung des *Ortsaspekts* nach Abbildung 2-12.

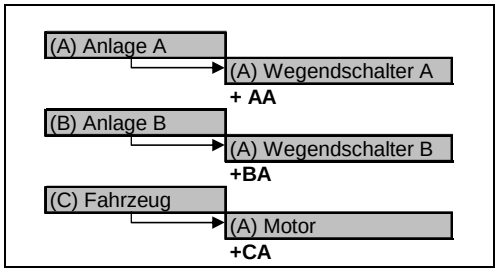

Abbildung 2-12: Ortsaspekt und Referenzkennzeichen

Der Ortsaspekt hat seine Bedeutung bei der Lokalisierung von Bauteilen in einer komplexen Anlage.

2.3.4 Abschlussbetrachtung Aspekte nach DIN EN 61346

Die Kennzeichnung von Bauteilen in einer Anlage nach DIN EN 61346 führt dazu, dass die Bauteile über drei verschiedene Referenzkennzeichen identifiziert werden können. Für das Beispiel des Fahrzeugs ergeben sich folgende Kennzeichen nach Tabelle 2-2.

DIN 61 346	Funktionsbezogenes Referenzkennzeichen	Produktbezogenes Referenzkennzeichen	Ortsbezogenes Referenzkennzeichen
Fahrzeug Motor	= CBA	- CBA	+ CA
Wegendschalter A	= CBAA	- ABAA	+ AA
Wegendschalter B	= CBAB	- BBAA	+ AB

Tabelle 2-2: Referenzkennzeichen der Aspekte für das Beispiel: Fahrzeug

Die Bauteile des Beispiels können jeweils über drei verschiedene Referenzkennzeichen identifiziert werden. In der gemeinsamen Projektarbeit führt das zu Verständigungsproblemen, weil jeder Beteiligte in seinen Beschreibungen (Lasten-/Pflichtenheft) nur das für ihn zutreffende Kennzeichen nutzt. Die DIN EN 61346 fordert deshalb, dass die Überführung der Referenzkennzeichnung von einem Aspekt in einen beliebigen Anderen möglich sein muss. Die heute auf dem Markt befindlichen CAD-Werkzeuge unterstützen nur den für das Fachgebiet relevanten Aspekt: 3D-CAD-Systeme besitzen eine objektorientierte Teilehierarchie gemäß Produktaspekt und E-CAD-Systeme arbeiten mit Funktions- und Ortsaspekt. Deshalb werden im Projekt durch die Beteiligten typischerweise „Übersetzungslisten“ beispielsweise im Microsoft Excel-Format geführt, die eine Überführung einer Referenzkennzeichnung von einem in den anderen Aspekt ermöglichen.

2.4 Planung in der Automatisierung

Der Automatisierung[1] kommt bei der Projektabwicklung, wie bereits angesprochen, nicht nur technisch, sondern auch bei der Terminsituation eine zentrale Bedeutung zu. Um einen Eindruck zu vermitteln, was Automatisierung im Anlagenbau bedeutet, wird im Folgenden die Automatisierung mit ihren Aufgaben für die Anlagen, ihren Planungswerkzeugen und deren Abhängigkeiten vorgestellt.

Die Anlagenautomatisierung wird durchweg für große industrielle Anlagen eingesetzt, bei denen umfangreiche und komplexe Automatisierungsaufgaben auszuführen sind. Bei den Automatisierungssystemen handelt es sich um speziell für diesen Einsatz konzipierte modulare Geräte und Einheiten, die sich durch Konfiguration, Projektierung und Programmierung für den gewünschten Einsatzzweck der Anlage anpassen lassen [10] [11]. Abbildung 2-13 zeigt eine nach diesen Gesichtspunkten aufgebaute Struktur eines Automatisierungssystems für die Anlagenautomatisierung. Die technische Anlage (Feldebene) ist über Bussysteme an die Speicherprogrammierbaren Steuerungen (SPS) der Basisautomatisierung angebunden. Die Automatisierungsaufgaben werden von

[1] **Anmerkung**: Der Begriff *Automatisierung* wird im Rahmen dieser Arbeit als Sammelbegriff für die Teilgebiete: Feldebene, Basisautomatisierung, Leitebene und Betriebsführungsebene benutzt. Damit wird der zunehmenden Dezentralisierung und Verteilung lokaler intelligenter Systeme auf der Feldebene Rechnung getragen. Die früher üblichen Grenzen zwischen Niederspannungsschaltanlage, den Systemen der Automatisierung (Prozessrechner) und der Bedienungsführung sind dadurch verwischt worden, dass die Systeme über Bussysteme miteinander vernetzt sind und die Niederspannungsschaltanlage sowie die Feldebene über ihre dezentralen Bauteile zu einem Teil des Automatisierungssystems geworden sind.

den SPS ausgeführt. Die SPS sind untereinander und mit den Systemen der Prozessleitebene über Bussysteme verbunden.

Während für die Hardwarekomponenten standardisierte Geräte, die in unterschiedlicher Konfiguration für die verschiedensten Anlagen verwendet werden können, eingesetzt werden, erfolgt die Programmierung der unterschiedlichen Systeme SPS, Leitebene und insbesondere die Betriebsführungsebene meist individuell für die jeweilige Anlage [10] [11]. Die Betriebsführungsebene ist in diesem Zusammenhang nur aus Gründen der Vollständigkeit aufgeführt, wegen ihrer ausschließlich individuell erstellten Programmierung für die jeweiligen Anlagen, wird sie bei den weiteren Betrachtungen in dieser Arbeit keine Rolle mehr spielen.

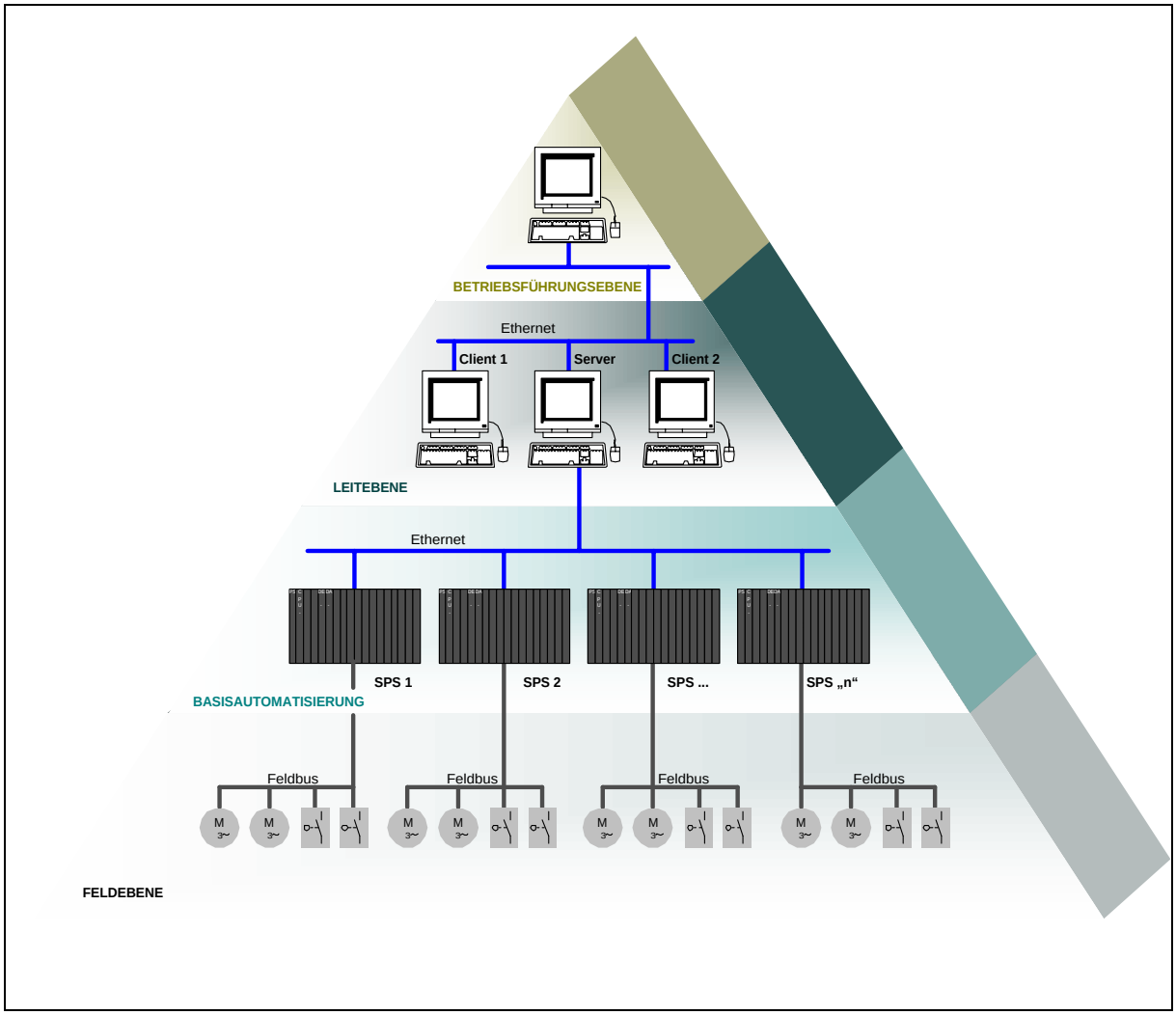

Abbildung 2-13: Hierarchie einer Automatisierungsstruktur

Die Automatisierung einer Anlage beinhaltet vier Teilgebiete, die zum Gesamtsystem *Automatisierung* zusammengeführt werden:

- Feldebene
- Basisautomatisierung
- Leitebene
- Betriebsführungsebene.

Auf den unterschiedlichen Ebenen werden verschiedene computerunterstützte Entwicklungssysteme benötigt und eingesetzt. Die nachstehende Tabelle 2-3 zeigt die auf den jeweiligen Ebenen bestimmenden Werkzeuge und die damit auszuführende Tätigkeit.

Hierarchieebene	Werkzeuge und Art der Tätigkeit	Anmerkung
Feldebene	Stromlaufpläne Elektro-CAD	Stromlaufpläne erstellen nach DIN EN 61346
Basisautomatisierung	SPS-Programmierung mit den Sprachen der Steuerung	Das Softwareentwicklungswerkzeug und die SPS-Hardware bilden eine (Entwicklungs-) Einheit.
Leitebene	Leitsystemprojektierung mit Graphik und Skriptwerkzeugen	Das Laufzeitsystem und die Editoren zur Erstellung bilden eine (Entwicklungs-) Einheit.
Betriebsführungsebene	Hochsprachenprogrammierung und Datenbankimplementierung	Verschiedene Hochsprachen und Datenbanksysteme sind kombinierbar.

Tabelle 2-3: Werkzeuge in der Automatisierung und deren Aufgabe

Auf Grund der technischen Entwicklung und des damit verbundenen Erfolges der Automatisierung bildeten sich auf den verschiedenen Ebenen Spezialwerkzeuge, die für die jeweilige Aufgabe hin optimiert wurden. Bis Mitte der 70er Jahre des letzten Jahrhunderts war die Stromlaufplanerstellung gleichzeitig die Lösung der Automatisierungsaufgabe, denn die eingesetzten Bauelemente waren entweder Hilfsschütze, Relais oder verdrahtbare Logikbaugruppen. Ungefähr 1975 begann mit der Einführung der Speicherprogrammierbaren Steuerung (SPS) die Trennung zwischen Stromlaufplan (Hardware) und der Automatisierungslogik (Software). Der Hersteller der SPS wurde zum Lieferanten des Entwicklungswerkzeuges zur SPS-Programmerstellung: dem Programmiergerät und seiner Software. Von der Einführung des Personal Computers (PC) konnte die Automatisierung in mehrfacher Hinsicht profitieren: Die Entwicklung preisgünstiger und leistungsstarker Programmeditoren war möglich, gleichzeitig standen für die SPS Hardware Mikroprozessoren zur Verfügung; höhere Programmiersprachen und relationale Datenbankmanagementsysteme (RDBMS) wurden für den PC entwickelt bzw. adaptiert. Mit der Einführung der vollgraphischen Darstellung auf dem

PC kamen die Schaltplanerstellung auf CAD-Systemen, sowie die Visualisierung und Bedienung von Prozessen hinzu [12].

Die oben aufgezeigten Entwicklungen liefen voneinander unabhängig und durch verschiedene Hersteller ab. Im Sinne der gemeinsamen Datenhaltung und des Datenaustausches handelt es sich dadurch um Insellösungen, siehe Abbildung 2-14. Jedes System benutzt sein eigenes proprietäres Format zum Speichern von Daten. Ab ungefähr 1995 begannen einzelne Hersteller ihre Produkte für die SPS-basierte Automatisierung als integrierte Lösungen für SPS-Programmierung und Leitsysteme zu harmonisieren, wodurch der Datenaustausch vereinfacht wurde. Das funktioniert aber nur solange, wie mit den Systemen eines Herstellers gearbeitet wird, müssen Systeme verschiedener Hersteller kombiniert werden, scheitert der integrierte Ansatz [12] [13].

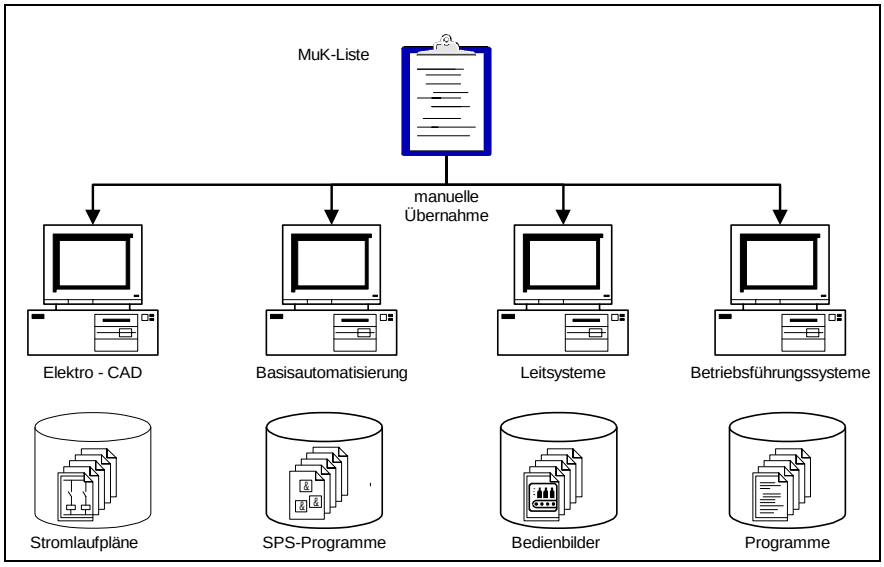

Abbildung 2-14: Werkzeuge zur Planung in der Automatisierung als Insellösungen

Für die Abwicklung des Projektes ist die fehlende gemeinsame Datenbasis hinderlich, da die einzelnen Arbeitsschritte in einer typischen Projektabwicklung aufeinander aufbauen. Der Hardwareplaner beginnt basierend auf der MuK-Liste mit der Stromlaufplanerstellung. Dabei werden die Adressen der SPS-Hardware vergeben. Anschließend beginnt der SPS-Programmierer die Erstellung des Programms und benötigt dazu die absoluten physikalischen Adressen aus der Hardwareplanung. Wegen des fehlenden Datenaustausches müssen diese Daten, die Symboliklisten, manuell übertragen werden. Danach kann die Projektierung des Leitsystems beginnen, weil das Leitsystem die Speicheradressen von Variablen (Tags) in der SPS kennen muss, um die Elemente der Bedienung und Beobachtung an die SPS anbinden zu können.

Den aufeinander aufbauenden Prozess der Projektierung in der Automatisierung zeigt Abbildung 2-15. Dieser Prozess bedeutete eine nacheinander ablaufende Projektierung; ein für alle Teilbereiche der Automatisierung gleichzeitig beginnender und parallel ablaufender Engineeringprozess (concurrent[1] engineering) ist nicht möglich.

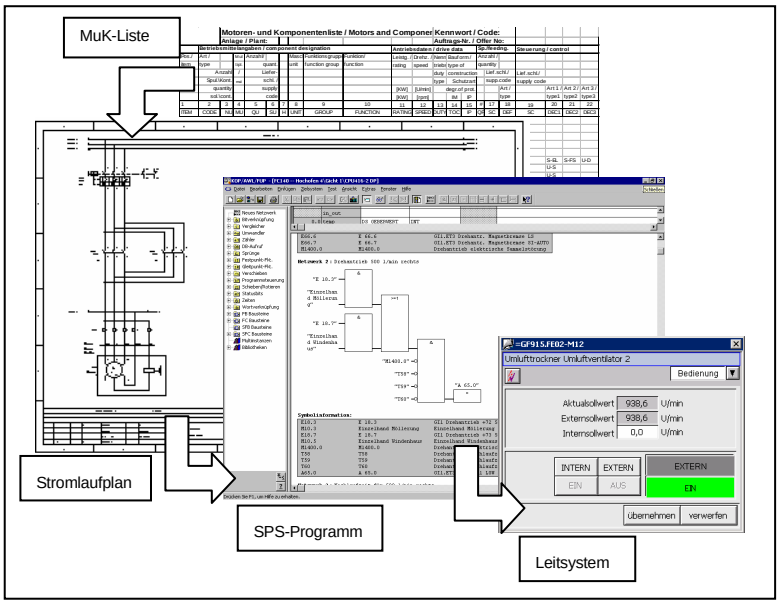

Abbildung 2-15: Ablauf der Projektierung in der Automatisierung: Von der MuK-Liste zum Leitsystem

Werden die vorgestellten Arbeitsschritte, die bei der Entwicklung einer typischen Automatisierungslösung erforderlich sind, als zeitliche Abfolge qualitativ in einen Terminplan nach Gantt aufgetragen, so entsteht eine Darstellung gemäß Abbildung 2-16.

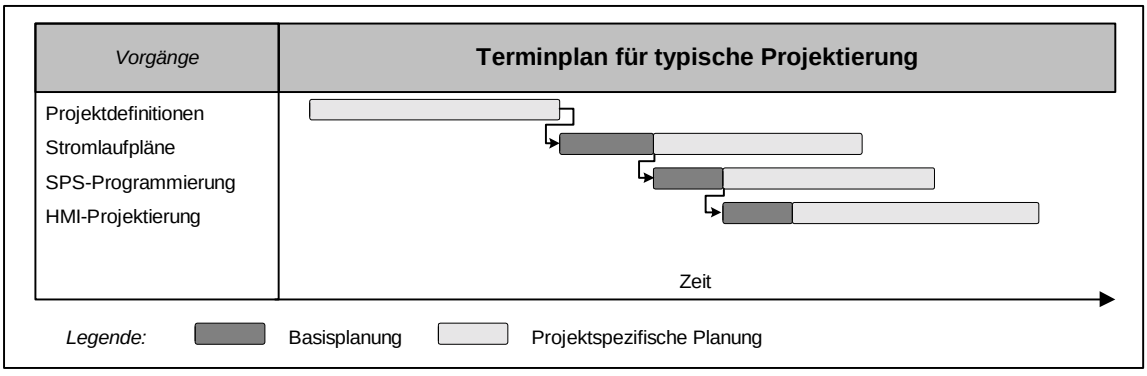

Abbildung 2-16: Qualitativer Terminplan für die typische Projektierung

[1] Concurrent (engl.) : gleichzeitig

Mit der Darstellung wird verdeutlicht, dass dem Beginn der Projektierung für die einzelnen Gebiete, eine Phase *Projektdefinition* vorgeschaltet ist. Während der Projektdefinition werden die Spezifikationen für die spätere Projektplanung zusammen mit dem Investor und dem Anlagenbau erarbeitet. Dabei handelt es sich um Planungsunterlagen wie Pflichtenheft, Funktionsbeschreibungen, Festlegung der Aspekte und Referenzkennzeichnung gemäß DIN, aber auch dem Erfassen von Werkvorschriften und deren Auswirkungen auf die Ausgestaltung der Projektierungslösung. Erst nach Abschluss der Definitionsphase beginnt die eigentliche Projektierung in den Teilgebieten, wie beschrieben. Jedes Teilgebiet beginnt, um mögliche Verzögerungen für die Nachfolgenden zu vermeiden mit dem Teil seiner Arbeit, auf den das nächste Gewerk aufbaut (Basisplanung) und wendet sich erst dann den projektspezifischen Lösungen zu.

2.4.1 Funktionsaspekt und E-CAD

Schaltungen sind in Stromlaufplänen nach funktionellen Gesichtspunkten gegliedert dargestellt. Auf die räumliche Lage und den mechanischen Zusammenhang der Bestandteile der jeweiligen Betriebsmittel wird in der Regel keine Rücksicht genommen. Die Kennzeichnung von Stromlaufplänen ist die konsequente Umsetzung der Funktionsaspekte nach DIN EN 61346[1]. Je nach Umfang der Schaltung sind mehrere Funktionsgruppen auf einem Blatt oder eine Funktionsgruppe auf mehreren zusammengehörigen Blättern, d.h. auf einer Blattgruppe dargestellt.

Dem Kennzeichnungsblock *Anlage* ist das Gleichheitszeichen (=) vorangestellt. Die Kennzeichnung entspricht vorwiegend dem funktionellen Aufbau, welchem auch die Gliederung der Stromlaufpläne entspricht. Ist eine Funktion auf mehrere Blätter verteilt dargestellt, so tragen die zusammengehörigen Blätter das gleiche Anlagenkennzeichen. Die alphanumerische Folge des Anlagenkennzeichens ist im Allgemeinen das Ordnungsschema für das Einordnen der Pläne. Das Anlagenkennzeichen, gleichzeitig Kennzeichen des Stromlaufplans, ist im Schriftfeld des Stromlaufplans eingetragen.

[1] Anm.: Für Stromlaufpläne galt DIN 40 719:1978, die eine Gliederung nach Funktions- und Ortsaspekt, unter der Bezeichnung Anlagen- und Ortskennzeichen, bereits vorschrieb. Mit Erscheinen der DIN EN 61346 wurde DIN 40 719 ungültig.

Der Kennzeichnungsblock ist in folgende 5 Abschnitte unterteilt:

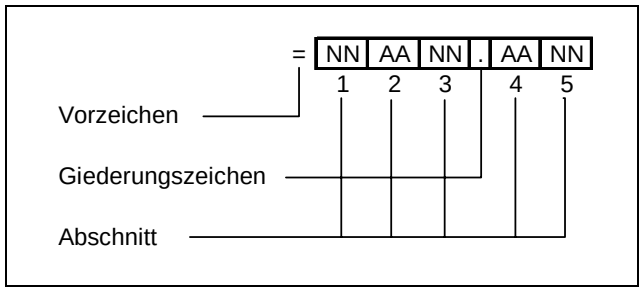

Abbildung 2-17: Kennzeichnungsblock für Stromlaufpläne

Die einzelnen Abschnitte sind durch eine abwechselnde Schreibweise von Ziffern und Buchstaben erkennbar. Innerhalb des Kennzeichnungsblockes werden höchstens 10 alphanumerische Datenstellen verwendet.

Die Anzahl der Datenstellen hängt von der Größe der Anlage ab. In der Praxis hat sich eine Gliederungstiefe von sechs Stellen vor dem Punkt (Abschnitt 1 - 3) als Referenzkennzeichen auch für komplexe Anlagen seit Jahren bewährt [14]. Nicht benötigte Datenstellen können entfallen, jedoch dürfen hierbei Abschnitte nicht übersprungen werden.

Die alphabetischen Datenstellen erhalten häufig eine bestimmte Bedeutung, die für die Anlage durch den Investor vorgegeben oder mit Erarbeitung des Pflichtenhefts abgestimmt wird. Die numerischen Datenstellen können reine Zählnummern sein oder eine bestimmte Bedeutung haben. Die alphabetischen und in der Regel auch die numerischen Datenstellen entsprechen den einzelnen Stufen der funktionellen Gliederung einer Anlage. Eine Stufe umfasst in der Regel gleichwertige Funktionen.

Der Kennzeichnungsblock beginnt links mit der größten und endet rechts mit der kleinsten funktionellen Einheit. Soweit zum Verständnis erforderlich, ist die gewählte Gliederung und die Bedeutung der Datenstellen in den Schaltungsunterlagen erläutert. Der Kennzeichnungsblock wird durch einen Punkt (.) als Gliederungszeichen zwischen den Abschnitten 3 und 4 unterteilt. Wie die Gliederung des Kennzeichnungsblocks zur Strukturierung einer Anlage eingesetzt wird, zeigt Abbildung 2-18.

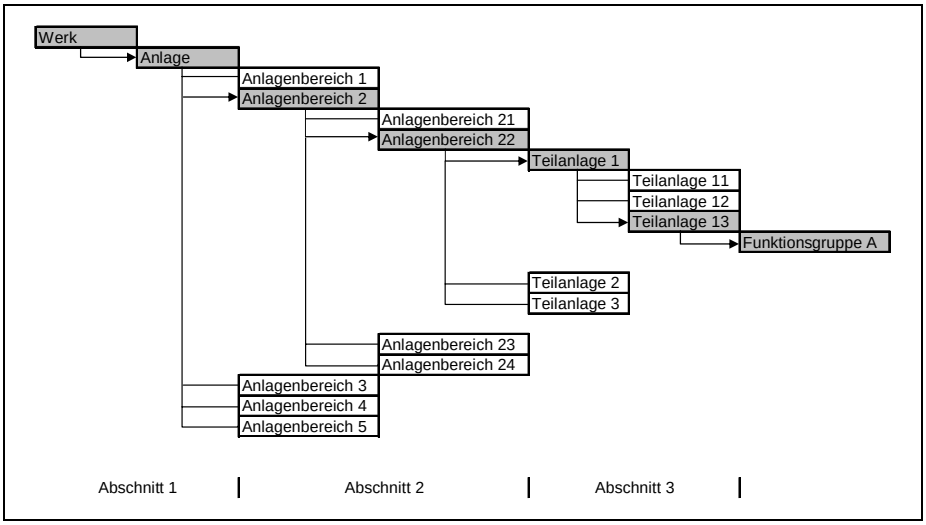

Abbildung 2-18: Strukturierung einer Anlage nach DIN EN 61346-1

Die Abschnitte 1 bis 3 links vom Gliederungszeichen enthalten Angaben über die aus Aggregaten, Funktionsgruppen usw. zusammengefügte Gesamtanlage.

Die Abschnitte 4 und 5 rechts vom Gliederungszeichen enthalten dann in der Regel Angaben über Gerätekombinationen, Aggregate, Funktionsgruppen (Leistungs-, Mess-, Regel- und Steuerkreis).

2.4.2 Steuerungsprogramm SPS

Steuerungssysteme auf Basis von Speicherprogrammierbaren Steuerungen sind, wie bereits dargestellt, noch recht junge Systeme in der Automatisierung. Feste, über Jahrzehnte hinweg bewährte und erprobte Strukturen, wie bei der Erstellung von Stromlaufplänen, sind bei SPS-Programmen daher noch nicht zu finden. Mit der Einführung der internationalen Norm IEC 1131 als DIN EN 61131 und insbesondere deren Teil 3 *Programmiersprachen* [15] wurde eine Harmonisierung der Programmiersprachen für SPS-Systeme angestrebt.

Seit der Einführung der Norm ist eine teilweise Harmonisierung der Programmiersprachen erfolgt. Ein Markt von unabhängigen Anbietern von SPS-Programmiersystemen hat sich dadurch nicht gebildet. SPS-Hersteller sind immer noch gleichzeitig Anbieter von SPS-Hardwarekomponenten und den zugehörigen SPS-Programmiersystemen, sie legen damit auch die Umsetzung der Norm in ihre Produkte (Hard- und Software) fest. Für den Anwender bedeutet das, dass er Programme oder Programmteile nicht von einem SPS-System zu einem Anderen übertra-

gen kann, weil die Implementierung der Programmiersprachen herstellerabhängig ist. Die Organisation PLCopen[1] arbeitet an der Harmonisierung der Programmiersprachen, damit eine gegenseitige Übernahme von einmal geschriebenen Programmen möglich ist. Jedoch ist die Anzahl der von PLCopen bisher nach DIN EN 61131-3 zertifizierten Programmiersystemen gering, wie Tabelle 2-4 zeigt.

Softwarewerkzeug	Sprache	Firma	Land
ACCON-ProSys	IL & ST	Deltalogic	Germany
Codesys	IL & ST	3S Smart Software Solutions	Germany
Concept V2.1	IL & ST	Schneider Automation	USA
Concept V2.5 SR2	FBD	Schneider Automation	Germany
DIApro / CAN	IL	Weidmüller Connext	Germany
ILD	IL	Welsch Software	Germany
ISaGRAF	IL	Altersys Europe	France
MELSEC MEDOC plus V2.31	IL	Mitsubishi Electric Europe	Germany
MULTIPROG wt	IL & ST	KW-Software	Germany
NAiS Control FPWIN Pro	IL	Matsushita Electric Works	Germany
openDK	IL	infoteam Software	Germany
openPCS	IL & ST	infoteam Software	Germany
PDS7	IL & ST	Nyquist	Netherlands
PUMA	IL & ST	KEBA	Austria
S7-SCL V4.02	ST	Siemens	Germany
Siemens Simotion Eng. Sytem Scout-standard	ST	Siemens	Germany
SELECONTROL CAP 1131	IL	Selectron	Switzerland
Soft Contol	IL	Softing	Germany
Sisteam Servicer (IEC 1131-3)	ST	TEAM	Spain
SUCOsoft S40	IL & ST	Moeller	Germany
S7-GRAPH V5.1	SFC	Siemens	Germany
Melsoft GX IEC Developer V. 4.0 SP2.1	IL	Mitsubishi	Germany
Melsoft GX IEC Developer V. 4.0 SP3	ST	Mitsubishi	Germany

Tabelle 2-4: Liste von PLCopen zertifizierten Programmiersystemen für SPS: Stand April 2003 [16]

Die Bedeutung der Kurzformen nach IEC der Spalte *Sprachen* ist in Tabelle 2-5 angegeben.

Sprache	Bedeutung
FBD	Funktionsbausteinsprache
IL	Anweisungsliste
LD	Kontaktplan
SFC	Schrittablaufsteuerung
ST	Strukturierter Text

Tabelle 2-5: Bedeutung der Abkürzungen in der Spalte *Sprachen*

[1] PLCopen: Hersteller- und Produktunabhängige Organisation zur Einführung internationaler Standards im Bereich der SPS, Zaltbommel NL

Von den Sprachen der Norm tritt der Kontaktplan *LD* unter den zertifizierten Sprachtypen gar nicht auf, die Funktionsbausteinsprache *FBD* und Ablaufsteuerungssprache *SFC* sind lediglich einmal vertreten. Die Sprachen *FBD* und *SFC* spielen in Deutschland eine große Rolle, wie die Werkvorschriften von Investoren zeigen [13]. Die Möglichkeiten zur Übernahme von Programmteilen über die Grenzen von Programmiersystemen hinaus sind, gemäß der vorstehenden Tabelle, zumindest bei den in Deutschland weit verbreiteten Sprachtypen, gering.

Bei der SPS-Programmierung haben sich trotzdem in der Praxis Grundstrukturen etabliert, wie eine Betrachtung des SPS-Systems SIMATIC STEP 7 zeigen soll. Durch Einführung von *Funktionsbausteinen* und *Funktionen* durch DIN EN 61131-3 hat sich eine Strukturierung des SPS-Programms etabliert, die sich typischerweise am Funktionsaspekt anlehnt [17].

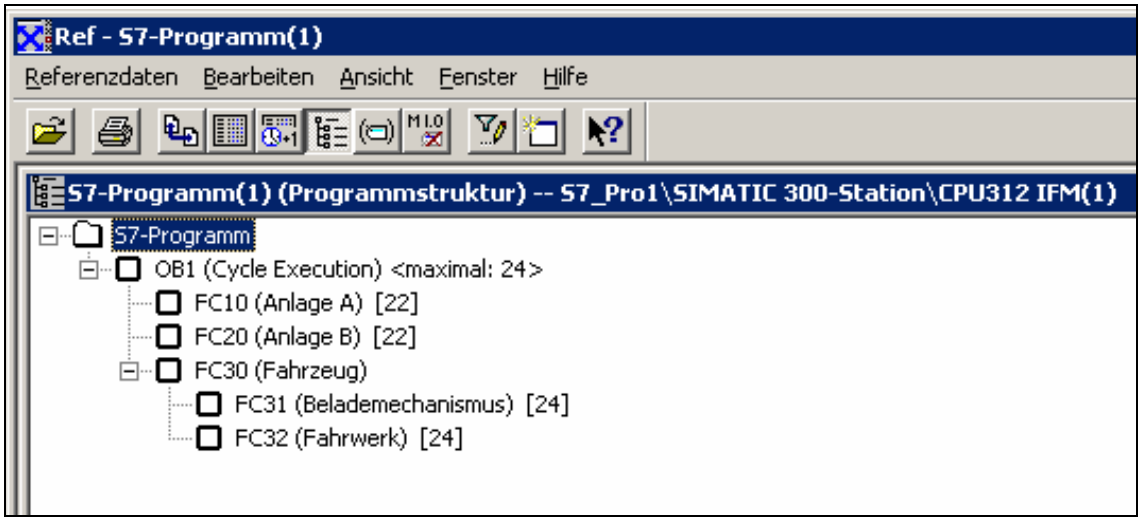

Abbildung 2-19: Programmstruktur in der SPS (SIMATIC S7)

Abbildung 2-19 zeigt eine nach diesen Grundsätzen aufgebaute Programmstruktur für das Beispiel *Fahrzeug*. Für die Teilbereiche werden Bausteine eingesetzt, die ihrerseits Bausteine von unterlagerten Funktionsteilen enthalten oder weitere Bausteine aufrufen. Auf diese Weise kann eine Struktur ähnlich dem Funktionsaspekt aufgebaut werden, wobei allerdings das SPS-System die maximale Schachtelungstiefe von Bausteinaufrufen, durch die Größe des erforderlichen Rücksprungstacks und des Lokaldatenbereichs, begrenzt.

Innerhalb eines Programmiersystems können *Funktionsbausteine* und *Funktionen* projektübergreifend genutzt werden. Für sich gesehen bietet die mehrfache Verwendung von Bausteinen in verschiedenen Projekten eine Produktivitätssteigerung gegenüber einer Neuprogrammierung. Von

Anwenderseite wird gefordert, dass auch größere Programmeinheiten, die ein elektro-mechanisches Grundsystem abbilden, wie in der MuK-Liste gezeigt, übernommen werden können [18].

Die Programmteile für ein elektro-mechanisches Grundsystem enthalten nicht nur Anweisungen, Funktionsbausteine und Funktionen zur Steuerung des Systems, sondern auch Schnittstellen beispielsweise zu:

- Betriebsartenumschaltungen (Hand-Automatik)
- Not-Halt-Kreisen
- Überlagerten Sequenz- oder Batchabläufen
- Störmeldesystemen
- Leitsystemen.

In den fehlenden Unterstützungsmechanismen zur Übernahme und Ablage von größeren Programmteilen liegen zurzeit die Defizite von SPS-Programmiersystemen.

2.4.3 Leitsysteme

Leitsysteme, auch HMI[1]-Systeme genannt, dienen der Anlagenbedienung und -beobachtung durch Anzeige von Anlagenzuständen, Prozessführung und Störbehandlung. Typische Vertreter von derzeit auf den Markt eingeführten Systemen, sind PC-basierte mit Windows 2000 aufgebaute Client-Server-Strukturen. Die Arbeitsweise ist dabei wie folgt: der Server ist mit der SPS-Ebene verbunden (Abbildung 2-13: *Hierarchie einer Automatisierungsstruktur*) und tauscht mit dieser die Informationen zur Bedienung und Beobachtung aus. Die Clients sind an den Server gekoppelt und dienen dem Bedienpersonal zur Anlagenführung.

Die elektro-mechanischen Grundsysteme werden üblicherweise in den Leitsystemen über Einblendfenster (Pop-Up[2]-Fenster) bedient, wie Abbildung 2-20 für eine Stellklappe zeigt. Jeder Typ von Grundsystem erhält einen eigenen Typ Fenster zur Bedienung und Anzeige. Das Fenster bezieht seine Daten für die Prozessführung von der unterlagerten SPS. Innerhalb der SPS wird auf den Programmteil des speziellen Aktors, für den das Pop-Up konfiguriert wurde, zugegriffen.

---

[1] HMI Abk. für **H**uman **M**achine **I**nterface (engl.) : Mensch-Maschine-Schnittstelle (Bedien- und Beobachtungssystem, Visualisierungssysteme)

[2] pop-up (engl.) : aufklappen, aufspringen

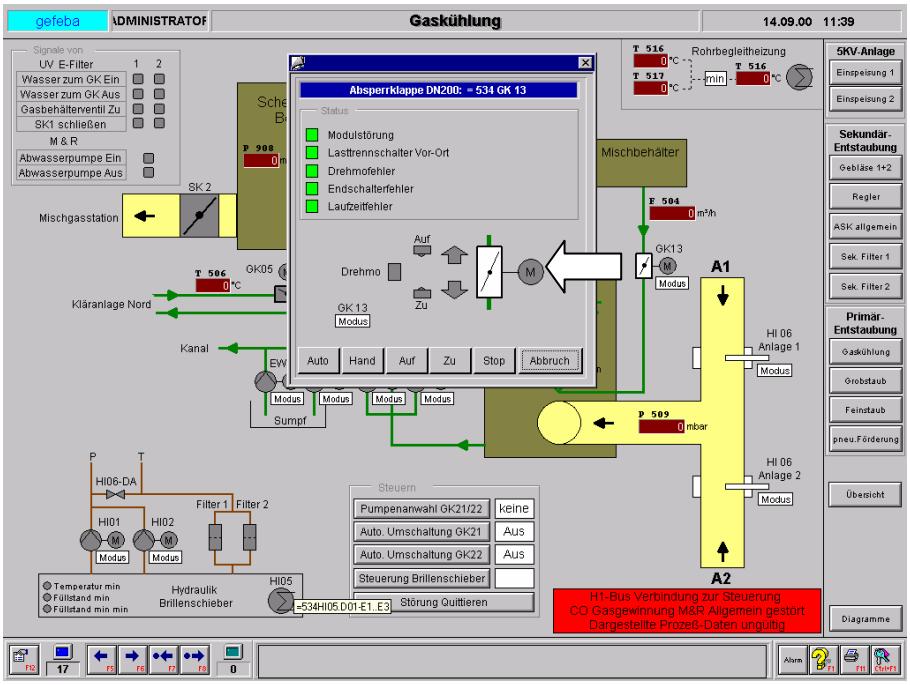

Abbildung 2-20: Typisches Anlagenbild mit Pop-Up-Fenster zur Bedienung eines Aggregates

Die Defizite der Leitsysteme sind dabei, fehlende Unterstützung beim Austausch von Projektierungsdaten zwischen HMI-Systemen unterschiedlicher Hersteller und fehlende Mechanismen zum Aufbau projektübergreifender Bibliotheken für Einblendfenster.

2.5 Typische Vorgehensweise der Projektplanung

Nach der Vorstellung der einzelnen Projektierungswerkzeuge und ihrer Aufgaben und Eigenschaften soll zum Gesamtverständnis die typische Vorgehensweise bei der Projektplanung diskutiert werden.

Die Abläufe des Engineeringprozesses in den einzelnen Fachgebieten der Planungsabteilungen von Maschinen- und Anlagenbauern sowie Automatisierungsausrüstern sind häufig über Jahre hinweg gewachsen. Die Wiederverwendung von Planungs- und Entwurfsunterlagen, insbesondere bei der Software, erfolgt in der Regel durch das Kopieren einzelner Dateien und Funktionen aus alten Projekten. Kopiert werden dabei typischerweise Stromlaufpläne der E-CAD-Systeme, Programmteile, Funktionsbausteine und Funktionen der SPS, sowie Einblendfenster für HMI-Systeme. Die kopierten Unterlagen werden für den Einsatz in der neu zu konstruierenden Anlage durch den Projekteur modifiziert [18].

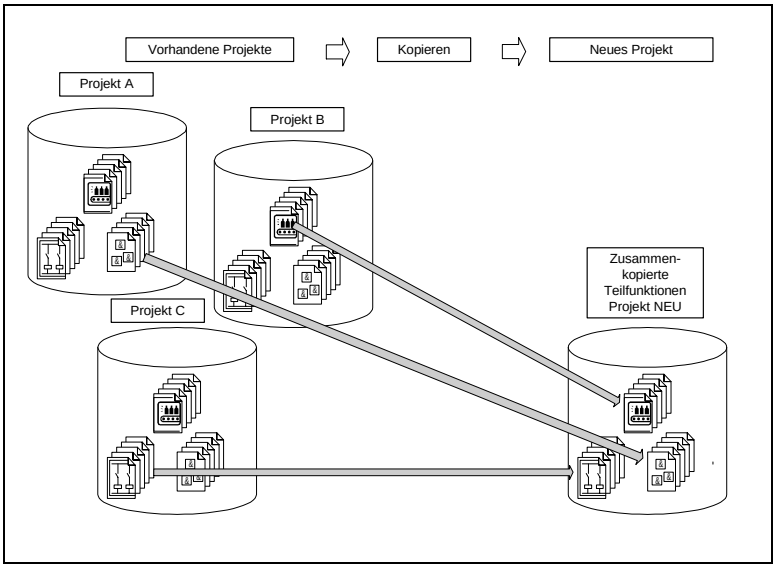

Abbildung 2-21: Typische Vorgehensweise bei der Projektierung [18]

Für sich gesehen bietet die typische Vorgehensweise durch Kopieren einzelner Funktionen, wie in Abbildung 2-21 illustriert, eine Produktivitätssteigerung, weil ganze Funktionsteile ohne Neuplanung übernommen werden können. Gleichzeitig weist diese Vorgehensweise jedoch gravierende Nachteile auf:

- *Die Engineeringkosten*: Es entstehen Mehrfachentwicklungen durch eine fehlende projektübergreifende oder unternehmensweite Abstimmung. Meistens greifen die Projektbeteiligten nur auf eigene Projekte zurück, weil sie mit Konstruktionen anderer Projektteams nicht ausreichend vertraut sind.
- *Die Qualität*: Fehler aus alten Projekten werden übernommen und Projektteile aus unterschiedlichen Projekten führen zu Inkonsistenzen im neuen Projekt.
- *Die Inbetriebnahme*: Mangelnde Qualität und stets andere uneinheitliche CAD- und Softwarestrukturen führen zu langen Inbetriebnahmezeiten.
- *Der Service und Gewährleistungen*: Auf Grund der fehlenden unternehmensweiten Standardisierung entstehen durch die notwendige projektspezifische Einarbeitung hohe Aufwendungen für das Servicepersonal [18].

Übernommene Funktionsteile sind mit den Daten, wie den Referenzkennzeichen, den technischen Daten und Beschriftungstexten aus der Ursprunganlage versehen; sie müssen für die Verwendung im neuen Projekt umbeschriftet oder umbenannt werden, bevor sie weiter benutzt werden können.

2.6 Derzeitige Problemfelder der Anlagenautomatisierung

Die Produkte für E-CAD, SPS-Software und HMI-Systeme zur Projektierung in der Anlagenautomatisierung decken mit ihren Funktionen die Anforderungen zur Anlagenplanung ab. Die Systeme unterstützen mit ihren bereitgestellten Funktionen nur den Projektierungsprozess im eigenen Teilgebiet. Sie kreieren zudem Datenformate, die nur im eigenen System verwendet werden können. Dadurch wird die Übernahme oder Kopplung zu Fremdsystemen sowie die Weiterverarbeitung in nachgeschalteten Projektierungswerkzeugen fast unmöglich gemacht. Für eine Engineeringunterstützung in der Automatisierung sind daher folgende Problemfelder aufzuarbeiten:

- *Harmonisierung:* Die Referenzkennzeichnung für die am Anlagenplanungsprozess beteiligten Fachgebiete unterscheidet sich und führt zu Kommunikationsproblemen, weil die Fachgebiete untereinander einen hohen Abstimmungsbedarf haben, um ein Bauteil eindeutig zu referenzieren. Der Koordinationsbedarf steigt mit der Anlagengröße, während bei einer kleineren Anlage mit wenigen Dutzend elektro-mechanischen Bauteilen eine verbale Beschreibung des Bauteils ausreichen mag, ist bei einer komplexen Anlage mit vielen hundert oder tausend Bauteilen eine Identifizierung ohne allgemeingültige Referenzkennzeichnung nicht mehr möglich

- *Standardisierung:* Die typische Planung in einem Unternehmen beschränkt sich vornehmlich darauf, Lösungen aus vorhandenen Anlagen durch einfaches Umkopieren auf neue Anlagen zu übernehmen. Eine systematische Sichtung und Katalogisierung der bereits vorhandenen Projektierungslösungen, um diese unternehmensweit zur weiteren Nutzung bereitzustellen, findet nicht statt.

- *Integration:* Mit Integration ist hier gemeint, dass die gefundenen Lösungen der einzelnen Planungswerkzeuge nicht aufeinander abgestimmt sind. Diese Lösungen müssen nicht nur in Entwicklungswerkzeugen Daten austauschen, wie beispielsweise Symboliklisten von SPS-Ein- und Ausgängen, sondern insbesondere in der fertigen Anlage also zur Laufzeit miteinander korrespondieren, beispielsweise beim Variablenaustausch zur Anlagenführung. Für die derzeitigen Planungssysteme stellt das ein unüberwindbares Problem dar. Selbst wenn SPS-Programmierung und HMI-Systeme für ein neues Projekt passende Lösungen in vorhandenen Projekten

finden, ist nicht gewährleistet, dass der Online-Datenaustausch der Altprojektierungen untereinander kompatibel ist.

- *Parallelität*: Die Unterstützung paralleler Entwicklungsprozesse in der Automatisierung stellt für die Systeme ein Problem dar. Der Informationsfluss der Projektierung geht basierend auf dem Modell der Automatisierungspyramide von der Feldebene (E-CAD) über SPS zu HMI und von dort zur Unternehmensleitebene, weil die einzelnen Projektierungsebenen Daten der Vorgängerebene als Grundlage ihrer weiteren Arbeit benötigen.

Im Folgenden soll ein Konzept für eine Engineeringunterstützung **dbEngineer** auf der Basis einer geeigneten Softwarearchitektur und Datenstruktur entwickelt werden, das alle aufgezeigten Problemfelder innerhalb eines Unternehmens fachgebietsübergreifend lösen kann. Hierbei soll der Schwerpunkt auf der Standardisierung und Katalogisierung von Projektierungslösungen liegen. Für das Zusammenspiel mit den am Projektierungsprozess beteiligten Softwarewerkzeugen, sollen die von den Herstellern unterstützten Mechanismen zum Einsatz kommen. Durch Nutzung der vorhandenen Kopplungsmöglichkeiten bleibt der Aktualisierungsaufwand bei einer Weiterentwicklung der Softwarewerkzeuge in einem vertretbaren Rahmen. Bevor aber das Konzept weiter diskutiert wird, sollen zunächst für das Konzept geeignete Softwarearchitekturen und Datenstrukturen vorgestellt werden.

3 GRUNDLAGEN FÜR SOFTWARESYSTEME

Für das Konzept des noch zu entwickelnden **dbEngineers** werden verschiedene Basistechnologien genutzt. Es handelt sich dabei um das Component Object Model[1] (COM) der Firma Microsoft, der allgemein nutzbaren Datenbankschnittstelle ADO[2], die ihrerseits auf COM beruht, sowie dem universellen Dokumentenformat Extensible Markup Language[3] (XML).

Wegen der grundlegenden Bedeutung für Funktion und Arbeitsweise des **dbEngineers** werden die genutzten Technologien im nachstehenden Kapitel erläutert.

3.1 Das Component Object Model (COM)

Bei COM [19] handelt es sich um ein sprachunabhängiges Software-Komponenten-Modell, das die Interaktion zwischen Softwarekomponenten und Windows-Anwendungen ermöglicht. Die besondere Bedeutung von COM besteht darin, dass durch dieses Modell klar definierte Schnittstellen die Kommunikation zwischen Komponenten, zwischen Anwendungen und zwischen Clients und Servern ermöglicht. Diese Schnittstellen bieten Clients die Möglichkeiten, von einer COM-Komponente Informationen über die zur Laufzeit unterstützten Funktionen abzufragen.

Eine Anwendung kann dabei auf die Schnittstellen von COM-Komponenten, die sich auf demselben Computer wie die Anwendung, oder auf einem anderen, über das Netzwerk erreichbaren Computer befinden, zugreifen. Im zweiten Fall (über das Netzwerk) geschieht dies mit einem Erweiterungsmechanismus des COM-Modells, der als Distributed[4] COM oder abgekürzt DCOM bezeichnet wird. Die gesamte COM/DCOM-Technologie beruht auf Windows und ist nicht plattformübergreifend [19].

Bei dem COM-Mechanismus handelt es sich einerseits um eine Spezifikation eines Modells als auch deren Implementierung. Bei der COM-Spezifikation werden die Erstellung von Objekten und die Kommunikation zwischen diesen Objekten definiert. Gemäß ihrer Spezifikation können COM-Objekte in unterschiedlichen Programmiersprachen erstellt sowie in unterschiedlichen Prozessräu-

---

[1] Component Object Model (engl.): Komponenten(-basiertes) Objektmodell

[2] ADO Abk. für: **A**ctiveX **D**ata **O**bjects (engl.): ActiveX Datenobjekte

[3] Extensible Markup Language(engl.): Erweiterbare Auszeichnungssprache

[4] Distributed (engl.): Verteilt; hier: verteilt auf verschiedene Rechner

men und auf verschiedenen Rechnern ausgeführt werden. Sind diese Objekte spezifikationskonform, können sie miteinander kommunizieren.

Das WIN32 Sub-System stellt die COM-Implementierung bereit. Dazu gehören eine Reihe von zentralen Diensten, die die Grundspezifikation von COM unterstützen. In der COM-Bibliothek sind eine Reihe von Standard-Schnittstellen enthalten, die die Hauptfunktionen eines COM-Objektes definieren. Weiterhin sind einige API[1]-Funktionen enthalten, mit deren Hilfe COM-Objekte erstellt und verwaltet werden können.

3.1.1 Elementare COM-Anwendungen

Um eine COM-Anwendung implementieren zu können, müssen folgende Elemente nach Tabelle 3-1 vorhanden sein.

COM-Schnittstelle	Die COM-Schnittstelle definiert die Art und Weise, in der ein Objekt seine Dienste für externe Clients bereitstellt. Ein COM-Objekt bietet eine Schnittstelle für jede Gruppe zugehöriger Methoden und Eigenschaften an. Die Schnittstelle ist in einer Typ-Bibliothek mit der Dateiendung .tlb, veröffentlicht.
COM-Server	Der COM-Server ist ein Modul (es kann sich dabei um eine .exe-, .dll- oder .ocx - Datei handeln), dass den Quelltext für das spezifizierte COM-Objekt enthält. Die Objektimplementierungen sind in den Servern enthalten.
COM-Client	Bei dem COM-Client handelt es sich um die Anwendung, die die Schnittstellen des Servers zum Abrufen der zuvor angeforderten Serverdienste nutzt. Die Clients wissen zwar, was sie über die Schnittstelle vom Server abrufen möchten, nicht aber, wie der Server seine Dienste intern bereitstellt (Kapselung).

Tabelle 3-1 Elemente einer COM-Verbindung [19]

3.1.2 COM-Schnittstellen

Die Kommunikation zwischen COM-Clients und den Objekten erfolgt über die COM-Schnittstellen. Bei den Schnittstellen handelt es sich um eine Gruppe von logisch und semantisch zusammen gehörenden Routinen, die die Kommunikation zwischen einem Anbieter des Dienstes (Serverobjekt) und seinen Clients ermöglicht. Standardmäßig wird eine COM-Schnittstelle wie folgt dargestellt:

[1] API: Abk. für **A**pplication **P**rogramming **I**nterface (engl.); Programmierschnittstelle (einer Anwendung)

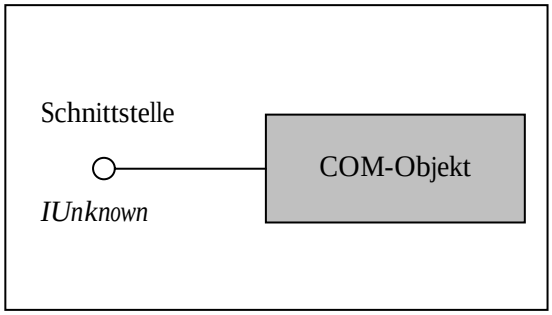

Abbildung 3-1: COM-Objekt [19]

Definitionsgemäß stellt jedes COM-Objekt die grundlegende Schnittstelle *IUnknown* (siehe Abbildung 3-1) bereit, über die die Clients abfragen können, welche Schnittstellen im Clientobjekt zur Verfügung stehen können. Diese Schnittstelle stellt die Möglichkeit dar, dem Client mitzuteilen, welchen Dienst der Server zur Verfügung stellt, ohne dass Implementierungsdetails darüber geliefert werden, wie und wo das Objekt diesen Dienst zur Verfügung stellt. Die wichtigsten Merkmale von COM-Schnittstellen sind dabei:

- Schnittstellen sind sprachunabhängig. Eine COM-Schnittstelle kann mit Hilfe jeder beliebigen Sprache implementiert werden, so lange diese Programmiersprache eine Struktur von Zeigern unterstützt oder mit ihr über einen Zeiger auf eine Funktion zugegriffen werden kann.
- Schnittstellen sind selbst keine Objekte, sondern sie stellen eine Möglichkeit zum Zugriff auf Objekte zur Verfügung.
- Schnittstellen entstehen immer durch Vererbung der grundlegenden Schnittstelle, wie *IUnknown.*
- Schnittstellen können von COM über Proxy-Server so umgeleitet werden, dass Aufrufe von Schnittstellenmethoden zwischen Threads, Prozessen und im Netzwerk laufenden Computern erfolgen können, ohne dass die Client- bzw. Serverobjekte diese Umleitung bemerken (DCOM).

3.1.3 Das Serverobjekt

Da ein Client bei COM nicht zu wissen braucht, wo sich ein Objekt befindet mit dem es sich verbinden will, weil der Client einfach dessen Objektschnittstelle aufruft, und die zum Aufrufen notwendigen Schritte automatisch vom COM durchgeführt werden, folgt hier ein kurzer Überblick

über die verschiedenen Möglichkeiten, wo ein Server sich befinden kann. Tabelle 3-2 zeigt die drei unterschiedlichen Typen von COM-Servern.

Servertyp	***Eigenschaften***
In-Process-Server	Eine Bibliothek (DLL) oder ActiveX, die im *selben Prozessraum* wie der Client ausgeführt wird. Der Client kommuniziert über direkte Aufrufe der COM-Schnittstelle mit dem In-Process-Server.
Out-of-Process-Server (bzw. lokaler Server)	Eine andere Anwendung (EXE), die in einem *anderen Prozessraum* , aber auf *demselben Computer* wie der Client ausgeführt wird. Der lokale Server kommuniziert über COM mit dem Client. Der Remote-Server verwendet DCOM zur Kommunikation mit Schnittstellen und dem Anwendungsserver.
Remote-Server	Eine DLL oder eine andere Anwendung, die auf einem *anderen Computer* als dem ausgeführt wird, auf dem der Client läuft. Beispiel: Eine Datenbankanwendung ist mit einem Anwendungsserver auf einem anderen Computer im Netzwerk verbunden.

Tabelle 3-2: Arten von COM-Servern [19]

Der Out-of-Process-Server und der Remote-Server sind besonders interessant, da sie in einem eigenem Prozessraum oder sogar einem eigenständigen Rechner zum Einsatz kommen können und damit völlig neue Möglichkeiten zur Nutzung von Softwarekomponenten eröffnen. Der Vollständigkeit halber seien die Wirkungsmechanismen aller Servertypen kurz angegeben.

Abbildung 3-2 zeigt, dass sich der Zeiger auf die Objektschnittstelle für In-Process-Server im selben Prozessraum wie der Client befindet, so dass COM auf die Objektimplementierung direkt zugreifen kann.

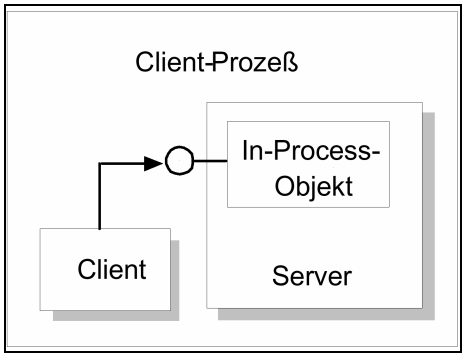

Abbildung 3-2: In-Process-Server [19]

Abbildung 3-3 zeigt die verschiedenen Zugriffswege bei Out-of-Process-Servern und Remote-Servern. Der Unterschied zwischen diesen beiden Servertypen besteht in der Art der Kommunikation zwischen den einzelnen Prozessen. Der Proxy-Server verwendet COM zur Kommunikation mit dem In- und Out-of-Process-Server, während bei der Kommunikation zu einem Remote-Computer verteiltes COM (DCOM) benutzt wird. Der Client merkt von der Umleitung des Zugriffs auf das Server-Objekt nichts.

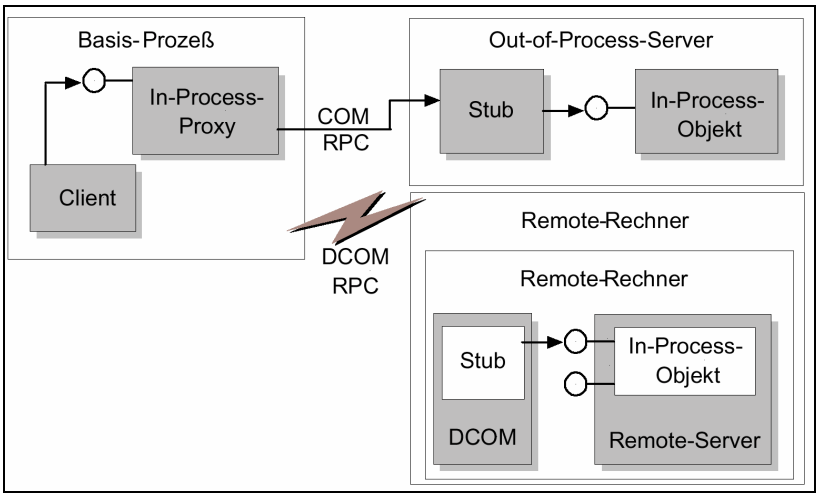

Abbildung 3-3: Out-Of-Process und Remote-Server [19]

Da es sich bei DCOM um einen Aufruf über das Netzwerk handelt, gelten die Vorschriften des ISO/OSI-Basisreferenzmodells für die Kommunikation [10]. Das Betriebssystem Windows und DCOM zusammen stellen der Applikation (Client) die Dienste gemäß ISO/OSI-Referenzmodell (ISO IS 7498) zur Verfügung. Als Protokoll wird dabei Remote Procedure Call[1] (RPC) benutzt und die Sicherungsschicht wird vom NTLM (NT LAN Manager) durchgeführt [19].

3.1.4 Vorteile der COM-Schnittstelle

- Sprachunabhängig: Clients können C, C++, Visual Basic, Borland Delphi, MS Office-anwendungen etc. sein.
- COM-Objekte können über Rechnergrenzen hinweg aufgerufen werden.
- Die Dienste gemäß ISO/OSI-Referenzmodell stellt COM/DCOM bereit.

[1] Remote Procedure Call (engl.): Funktionsaufruf über Netzwerke, definiert von der **O**pen **S**oftware **F**oundation (OSF) als **D**istributed **C**omputing **E**nvironment (DCE)

3.1.5 Beispiel der COM-Nutzung: ActiveX Data Objects (ADO)

Auf Basis der COM-Technologie entwickelte Microsoft den Mechanismus eines universellen Datenzugriffs, der es ermöglicht, auf unterschiedliche Datenbanken und Daten mit einem einheitlichen Verfahren zugreifen zu können. Die Gesamtstrategie eines universellen Datenzugriffs wurde von Microsoft unter dem Namen *Universal Data Access*[1] (UDA) eingeführt. Dabei implementieren die COM-Objekte der *ActiveX Data Objects*[2] (ADO) den clientseitigen Zugriff auf die Daten. Auf der Serverseite gehören z.B. der Microsoft SQL Server, Microsoft Exchange Server oder die Windows 2000 Index Service[3] zu den Bestandteilen der UDA-Strategie [19][20].

Ein Beispiel für die Nutzung von COM ist die von Microsoft geschaffene Zwischenschicht für den Datenzugriff ADO. Die Aufgabe von ADO ist es, gegenüber den Anwendungen eine Abstraktionsebene zu bilden, die es ermöglicht, mit den gleichen Methodenaufrufen der ADO-Objekte unterschiedlichste Datenbanken und Daten zu bearbeiten, ohne dass sich die Anwendung um die Implementierung, Eigenschaften oder Standort der Daten kümmern muss. Seit den Versionen der Windows 2000-Familie sind ADO und die ADO unterstützende Schicht *Object Linking and Embedding Database*[4] (OLE-DB) ein Teil des Betriebssystems. Für ältere 32Bit-Versionen von Windows können die Objekte nachinstalliert werden.

Abbildung 3-4 illustriert das Zusammenwirken der unterschiedlichen Schichten: Der Datenbankhersteller oder ein Drittanbieter liefert einen nativen Datenbanktreiber, den OLE DB Provider. Der OLE DB-Provider wird von der Schicht OLE DB benutzt, um direkt auf die Informationen (Datenbank B und Excel) zugreifen zu können. Die Schnittstelle zwischen OLE DB und dem OLE-DB Provider ist von Microsoft spezifiziert. Die Abbildung zeigt, dass OLE DB weiterhin die Treiber der Vorgängertechnologie *Open Database Connectivity*[5] (ODBC) einbinden und nutzen kann (Datenbank A). Die UDA-Strategie beschränkt den Zugriff der Anwendung nicht auf Datenbanken, sondern sie gilt für Datenquellen im Allgemeinen beispielsweise für Tabellenkalkulationen oder, wie oben bereits erwähnt, dem Microsoft Exchange Server.

---

[1] Universal Data Access (engl.): Universeller Datenzugriff

[2] Anm.: ActiveX sind In-Process-Server

[3] Index Sevice (engl.): Indexverzeichnisdienst (Anm.: Teil des Betriebssystems)

[4] Object Linking and Embedding Database: (engl.): Objektzugriff und Einbettung von Datenbanken

[5] Open Database Connectivity (engl.): Offener Standard für Datenbankverbindungen

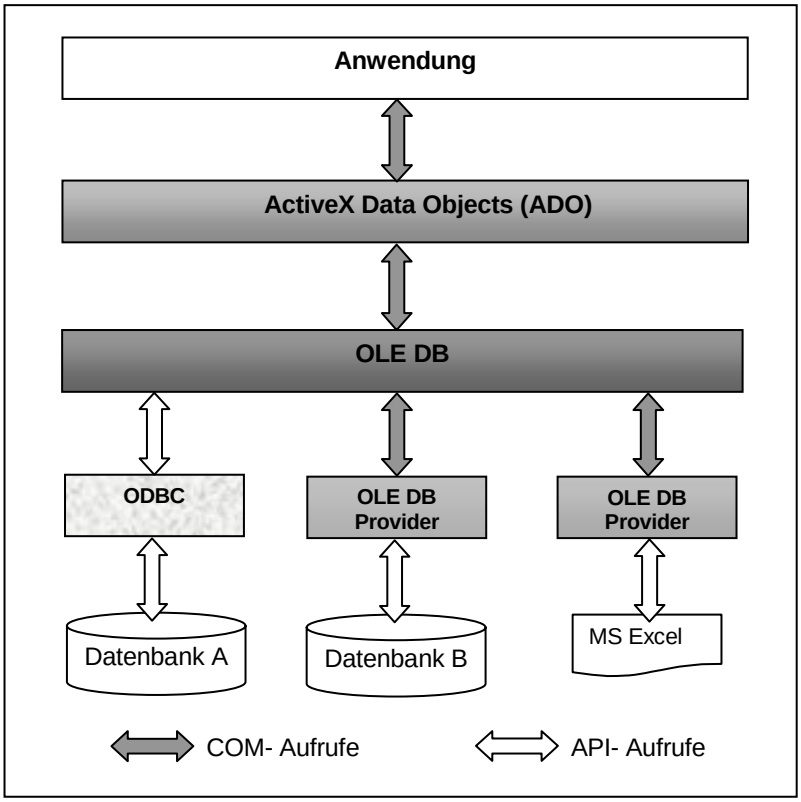

Abbildung 3-4: Anwendung, ADO, OLE DB und OLE DB-Provider im Zusammenspiel (Datenbank B und Excel)

Die über OLE DB liegende Schicht ADO stellt der Anwendung Objekte zum Zugriff auf die Daten zur Verfügung. Das Zusammenspiel von ADO und OLE DB ist für den Programmierer transparent, so nimmt das ADO-Objekt *Connection*[1] die Information über die zu öffnende Datenverbindung entgegen und reicht sie für den Programmierer verborgen an OLE-DB weiter. ADO könnte auch als das Programmierinterface von OLE-DB bezeichnet werden. [20].

Abbildung 3-5 zeigt die Erstellung einer Datenverknüpfungseigenschaft, wobei diese Informationen ADO entweder als Verbindungsstring oder als Datenverknüpfungsdatei übergeben werden können. Für die Client-Anwendung bietet die Abstraktion durch die ADO-Komponenten den Vorteil, dass das Datenbankmanagementsystem (DBMS) gewechselt werden kann, ohne dass eine Programmanpassung erforderlich ist. In der Praxis heißt das: ein einmal erstelltes Programm kann ohne Änderungen bei Anwendern, die unterschiedliche Datenbankumgebungen und Serverkonzepte besitzen, betrieben werden. Bei einem Wechsel des DBMS reicht es, eine neue Datenverknüp-

[1] connection (engl.): Verbindung

fungsdatei zu erstellen, die auf die geänderte Datenbank verweist. Beim nächsten Aufruf des Programms wird die gewechselte Datenbank angesprochen.

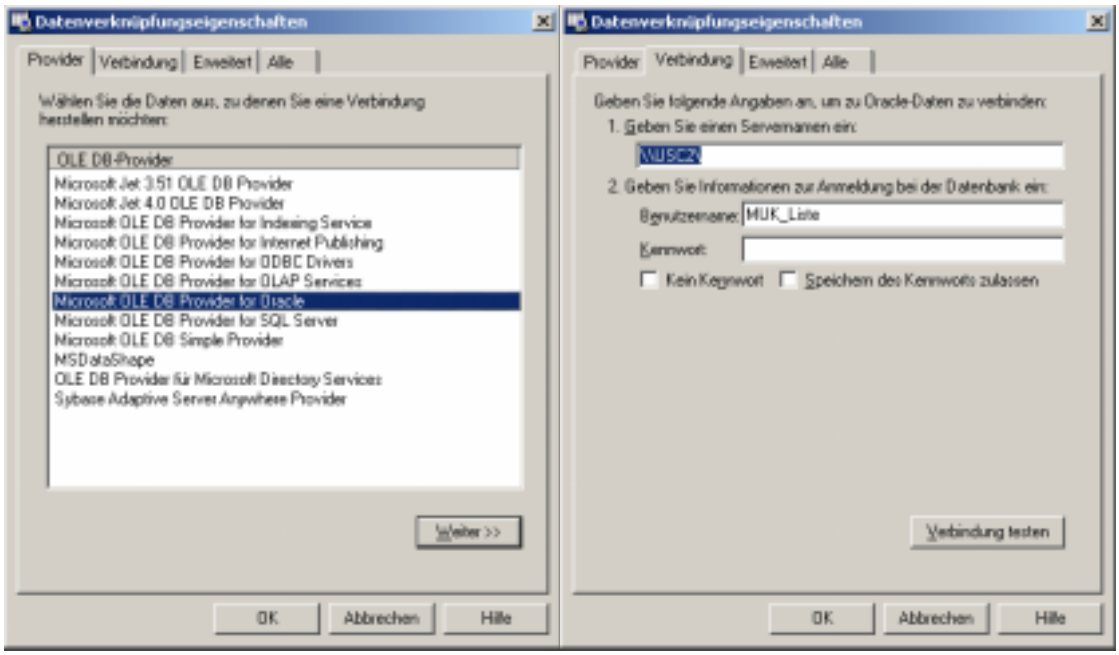

Abbildung 3-5: Erstellung des Verbindungsstrings oder Datenverknüpfungsdatei für eine ADO-Connection

Der Client kann mit Hilfe von ADO problemlos in einer SQL-Abfrage die Datenbanktabellen mehrerer unterschiedlicher DBMS gleichzeitig ansprechen. Mit der ADO-Technologie können beispielsweise Informationen aus zwei verschiedenen CAD-Systemen, die unterschiedliche DBMS besitzen, kombiniert werden.

Für einen funktionsfähigen Zugriff auf die Daten müssen auf dem Client installiert sein:

1. Anwendung
2. ADO
3. OLE DB
4. Nativer OLE DB-Provider.

Der Standort der Daten ist nicht festgelegt, sie können entweder lokal mit auf dem Client Rechner liegen oder auf einem Remote – Datenbankserver ablegt werden. Den Zugriff auf Daten übernimmt der OLE DB-Provider, dessen Konfiguration auf den Standort verweist [19].

3.1.6 Die Erweiterung zu COM+

Mit dem Erscheinen von Windows 2000 erfuhr der COM/DCOM-Mechanismus eine Erweiterung zu COM+. Die COM+ Mechanismen wurden gleichzeitig zu einem Bestandteil des Betriebssystems von Windows.

Für die Administration von COM+ Anwendungen wurden die Komponentendienste eingeführt. Damit ist eine zentrale Verwaltung und insbesondere Rechtevergabe für COM+ Anwendungen möglich. Für jede COM+ Anwendung, für jedes Objekt, jedes Interface und sogar für jede Interface-Methode können Zugriffsregeln definiert werden. Über die COM+ Roles[1] stellt das Betriebssystem sogar eine Gruppenverwaltung für die Zugriffsrechte zur Verfügung, die vollständig in das Sicherheitssystem von Windows 2000 integriert ist, wie Abbildung 3-6 zeigt.

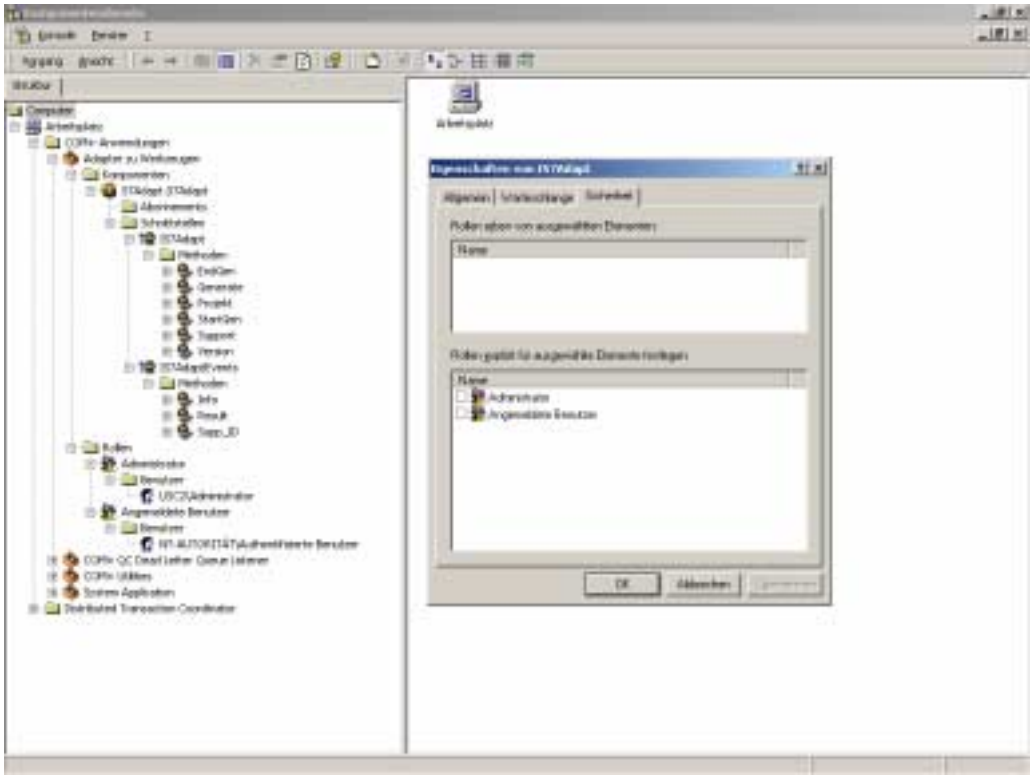

Abbildung 3-6: Rechtevergabe für eine COM+ Anwendung unter Windows 2000

Für die vollständige Beschreibung aller Änderungen und Erweiterungen, die mit Einführung von COM+ verbunden sind, sei auf die Standardliteratur verwiesen [19][21].

[1] Roles (engl.): Rollen

3.2 Extensible Markup Language (XML)

Die Extensible Markup Language ist ein einfaches, sehr flexibles Textformat, das von SGML[1] (ISO 8879) [22] abgeleitet wurde. Die Auszeichnungssprache XML wurde im Jahr 1998 als offener Standard vom W3C[2] vorgestellt und seitdem dort weiter entwickelt und verwaltet. Ursprünglich wurde XML entwickelt, um die auftretenden Probleme beim Veröffentlichen umfangreicher elektronischer Dokumente (Electronic Publishing) zu lösen. Heutzutage spielt XML eine zunehmend größer werdende Rolle im Austausch von Daten im Web und im Allgemeinen [23].

Leistungsmerkmale von XML sind:

- Mit XML kann fast jede Art von Information gespeichert und organisiert werden.
- Als offener Standard ist XML nicht an die Spezifikation eines einzelnen Unternehmens oder einer bestimmten Software gebunden.
- Mit Unicode[3] als Standard-Zeichensatz unterstützt XML eine Vielzahl von Schriften und Symbolen sowie Sonderzeichen.
- XML stellt Methoden zur Überprüfung der Qualität eines Dokumentes zur Verfügung, darunter Regeln für die Syntax.
- Mit seiner klaren, einfachen Syntax und seinen eindeutigen Strukturen, ist XML problemlos sowohl vom Menschen als auch von Programmen zu lesen und zu analysieren.

Die Auszeichnungssprache XML soll im Folgenden nur soweit vorgestellt werden, wie es für das Verständnis des in dieser Arbeit zu entwickelnden Datenaustauschformats erforderlich ist. Für weitergehende Informationen sei auf die Veröffentlichungen von W3C und die Standardliteratur verwiesen [23][24].

3.2.1 Konzepte von XML

Trotz seines Namens ist XML selbst keine Auszeichnungs- oder Markup-Sprache, es ist lediglich eine Zusammenfassung von Regeln, zur Schaffung von Markup-Sprachen. Markups selber sind

---

[1] **S**tandard **G**eneralized **M**arkup **L**anguage (SGML) (engl): Genormte verallgemeinerte Auszeichnungssprache

[2] **W**orld **W**ide**W**eb **C**onsortium (W3C): Freiwilliger Zusammenschluss von Firmen zur Förderung des Internets

[3] Unicode: Ein Standard-Zeichensatz mit 16-Bit Kodierung, mit dem versucht wurde, die Zeichen aller wichtigen Schriftsysteme der Welt zu berücksichtigen [24].

Informationen, die einem Dokument hinzugefügt werden, um auf bestimmte Weise dessen Bedeutungsgehalt zu erweitern. Die Auszeichnung geschieht dadurch, dass die einzelnen Teile gekennzeichnet werden und festgelegt wird, wie diese zueinander in Beziehung stehen [24].

Die Markup-Sprache selber ist eine Menge von Symbolen, die im Text des Dokumentes platziert werden, um die einzelnen Teile dieses Textdokumentes zu benennen und gegeneinander abzugrenzen. XML definiert keine Markups oder deren Bedeutung. Es ist vielmehr die Aufgabe des Anwenders für seine Zwecke Markups zu kreieren und deren Bedeutung zu veröffentlichen.

Da die Auswertung und Verarbeitung eines XML-Dokumentes elektronisch erfolgen soll, benötigt das auszuwertende Programm Informationen über den Beginn und das Ende von bestimmten Informationen bzw. Abgrenzungen von Informationsteilen gegeneinander. Durch die Markups erhält das auswertende Programm die so benötigten Informationen. Durch den Einsatz von Markups ist es möglich, ein Dokument zu strukturieren.

Die Markup-Sprache stellt dazu unter anderem zur Verfügung:

- Prolog
- Begrenzer
- Schachtelungen

Die Elemente werden im Kapitel *Syntax von XML* erläutert.

3.2.2 Syntax von XML

Der Inhalt eines XML-Dokumentes besteht aus Text, der mit Markup-Symbolen durchsetzt ist.

Prolog

Ein XML-Dokument beginnt immer mit einem Prolog, der spezielle Informationen über das Dokument beinhaltet. Im einfachsten Fall besagt der Prolog lediglich, dass es sich um ein XML-Dokument handelt und welche XML-Version benutzt wird.

```
<?xml version="1.0" standalone="yes"?>
```

Abbildung 3-7: Der XML-Prolog (Beispiel)

Im Beispiel Abbildung 3-7 handelt es sich um ein Dokument, das mit XML Version 1.0 erstellt wurde. Die Information *standalone*[1] = "*yes*" besagt, dass dieses Dokument keine weiteren zusätzlichen Ressourcen referenziert oder benötigt und für sich allein steht.

Begrenzer

Die Begrenzer oder Tags, zu erkennen an den spitzen Klammern <> (siehe Abbildung 3-8) und die von ihnen umschlossenen Namen, begrenzen und bezeichnen Teile des Dokumentes und fügen weitere Informationen hinzu, die zur Definition der Struktur des Dokumentes beitragen. Der Text zwischen den Tags ist der Inhalt des Dokumentes. Die Tags werden immer paarig gesetzt. Das erste Tag <Text> markiert den Beginn einer Kennzeichnung und dessen Negation, dargestellt durch </Text> markiert dessen Aufhebung bzw. Ende.

Die erste Regel für XML-Dokumente besagt, dass ein Element, das Text und Textelemente enthält, einen Start- und einen Endtag haben muss (Abbildung 3-8).

```
<Text>XML ist eine Auszeichnungssprache</Text>
```

Abbildung 3-8: Beispiel für den Einsatz von Begrenzern

Schachtelung

Bei der Schachtelung von Elementen ist darauf zu achten, dass sich diese nicht überlappen, sondern nur ineinander geschachtelt platziert werden. Bei richtiger Anwendung der Tag-Elemente können hierarchische oder baumstrukturierte Dokumente aufgebaut werden, die Beziehungen von Elementen untereinander ausdrücken.

```
<?xml version="1.0" standalone="yes"?>
<Gruppe>
  <Mitglied>Ernst</Mitglied>
  <Mitglied>Paul</Mitglied>
  <Mitglied>Hugo</Mitglied>
</Gruppe>
```

Abbildung 3-9: Strukturierung mit Begrenzern

[1] standalone (engl.): allein dastehend; unabhängig

Abbildung 3-9 stellt dar wie mit einem Begrenzer <Gruppe> die nachfolgenden Elemente zu einer Gruppe zusammengefasst werden können. Werden die Begrenzer geschachtelt, entsteht eine fortgesetzte Strukturierung, wie in Abbildung 3-10 gezeigt.

```
<?xml version="1.0" standalone="yes" ?>
<Abteilung>
  <Gruppe>
    <Mitglied>Ernst</Mitglied>
    <Mitglied>Paul</Mitglied>
    <Mitglied>Hugo</Mitglied>
  </Gruppe>
  <Gruppe>
    <Mitglied>Felix</Mitglied>
    <Mitglied>Fritz</Mitglied>
    <Mitglied>Sepp</Mitglied>
  </Gruppe>
</Abteilung>
```

Abbildung 3-10: Geschachtelte Gruppierung mit Begrenzern

Die Struktur <Abteilung> besteht aus 2 Strukturen <Gruppe>, diese Schachtelung entspricht einer aufgelösten 1:N-Beziehung, wie sie aus Relationalen Datenbankmodellen bekannt ist. Die gezeigten Beispiele in Abbildung 3-8 bis Abbildung 3-10 sind mit einem einfachen Texteditor erstellt worden und können z.B. im Microsoft Internet Explorer angezeigt werden.

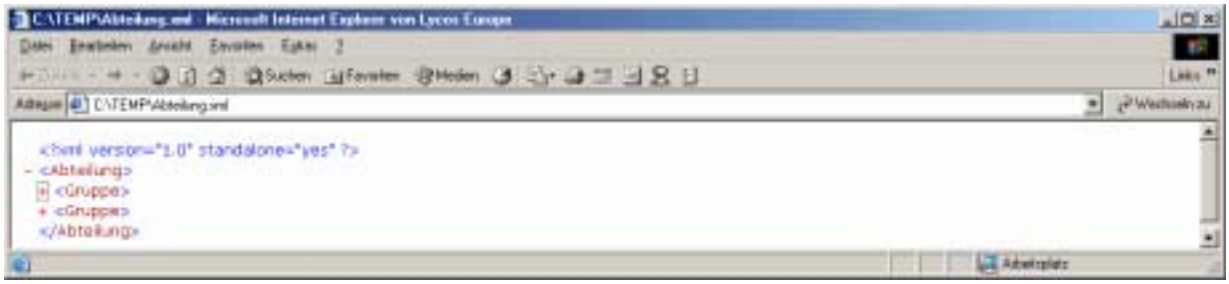

Abbildung 3-11: Darstellung der Gruppierung im Microsoft Internet Explorer

Wie Abbildung 3-11 zeigt, kennt der Explorer keine besondere Sicht auf die XML-Datei. Der Explorer bildet die Struktur als Knoten ab, die durch Anklicken der Pluszeichen (+) erweitert oder der Minuszeichen (-) reduziert werden können.

Die erweiterte Sicht mit allen Details der Struktur zeigt die nachstehende Abbildung 3-12. Der Browser setzt den Prolog und die Tags gegenüber den Textinhalten zur Erhöhung der Lesbarkeit farblich ab.

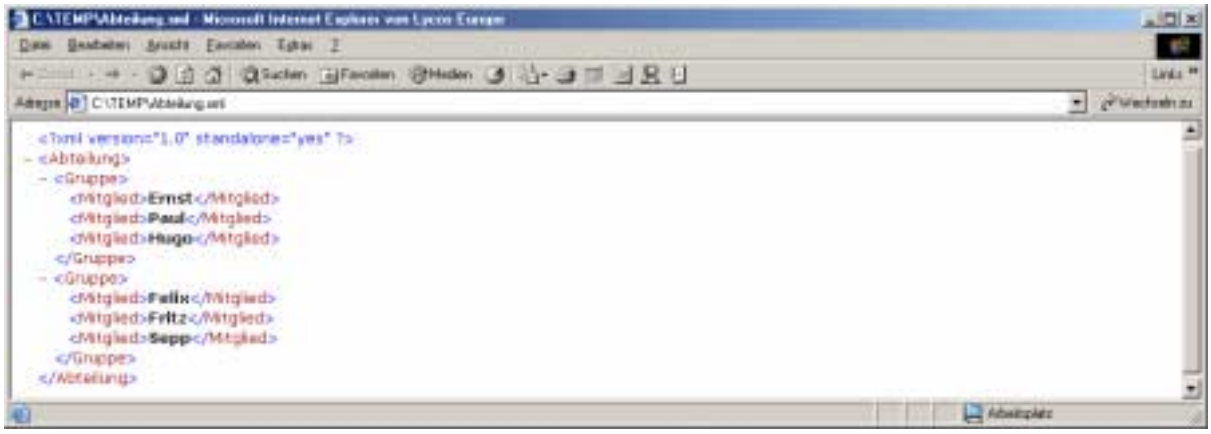

Abbildung 3-12: Detailansicht der Gruppierung im Microsoft Internet Explorer

Mit dem obigen XML-Beispiel konnte gezeigt werden, dass nicht nur die Textinformation (Name des Mitgliedes) in der XML-Datei abgelegt wird, sondern auch deren Beziehung untereinander. Die Gruppenzugehörigkeit ist in der Datei eindeutig beschrieben.

Es muss beachtet werden, dass die XML-Datei keinerlei Informationen über die Präsentation ihrer Daten liefert. Das bedeutet, es ist Aufgabe des öffnenden Programms die Daten formatiert darzustellen. XML-Dateien nehmen nur die Daten auf. Es besteht für die Präsentation die Möglichkeit eine XML-Datei mit weiteren Ressourcen wie Stylesheets[1] zu kombinieren, um das Erscheinungsbild des XML-Dokuments zu spezifizieren. Für den Umgang mit Stylesheets sei auf die Standardliteratur verwiesen [24].

3.2.3 XML als Datenaustauschformat

Da XML die Schachtelung und Kaskadierung von Elementen und Beziehungen untereinander durch geeignete Strukturierung ausdrücken kann, ist XML ein hervorragend geeignetes Instrument, zum Austausch von Datenbankinhalten. Durch die Nutzung der Strukturierung ist es möglich, in einem Dokument auch 1:N Beziehungen zwischen Datenbanktabellen zu übertragen und auszudrücken.

Viele Programme wie Tabellenkalkulationen, Programmiersysteme wie z.B. Borland Delphi oder Datenbankzugriffskomponenten, wie die im Kapitel 3.1.5 vorgestellten ADO-Komponenten,

[1] Stylesheet (engl.): *hier* Formatierungsanweisungen (entweder als zusätzliche Datei oder im XML-Dokument verpackt)

verfügen über einen integrierten XML-Prozessor, um Datenbankinformationen in XML-Dateien abzuspeichern, bzw. Dateninformationen aus XML-Dateien zu importieren.

Obwohl XML geeignet ist, Datenbankinhalte zu speichern, ist XML aufgrund seines Overheads an Informationen nicht dazu geeignet, als Speicherformat für Datenbanken zu dienen. Hier sind Relationale oder Objektorientierte Datenbanksysteme mit ihren optimierten Speicher- und Verwaltungsmechanismen deutlich überlegen [24].

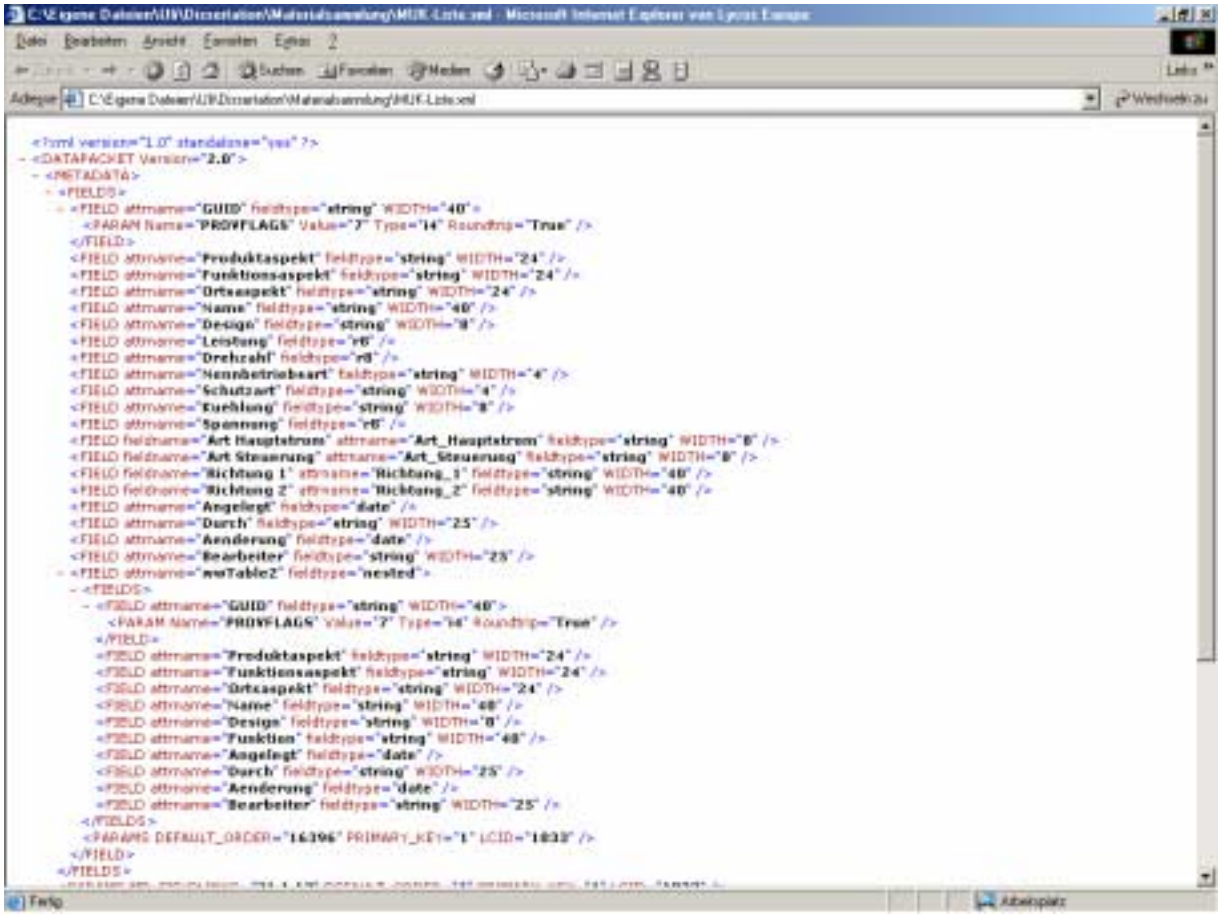

Abbildung 3-13: Datenbankbeschreibung einer 1:N-Relation in einem XML-Dokument

Abbildung 3-13 zeigt einen Ausschnitt einer Datenbank, die mit XML übertragen wurde. Dabei handelt es sich um zwei Datenbanktabellen, die 1:N verknüpft sind. Dargestellt ist, dass bei der Abspeicherung in XML, beginnend mit dem ersten Auftreten des Tags <*Fields*>, die Master-Datenbanktabelle beschrieben wird, wobei jedes einzelne Attribut der Tabelle mit seinen Eigenschaften und mit seiner Domain beschrieben ist, und mit einem weiteren Tag <*Fields*> beginnt die

Definition der verschachtelten Tabelle *wwTable2* (1 : N Relation). Erst dahinter folgen die Dateninhalte der Datentabelle (nicht mehr angezeigt).

XML-Dokumente enthalten nicht nur die Nettodaten, sondern können, wie in Abbildung 3-13 gezeigt, ihre Felder und Feldtypen beschreiben sowie ihre Dateninformationen strukturieren. Mit diesen Eigenschaften sind sie hervorragend geeignet, den Datenaustausch, wo es um begrenzte Informationsmengen zwischen zwei unterschiedlichen Systemen geht, zu vereinfachen. XML-Dateien können beispielsweise als Anhang einer E-Mail verschickt werden oder weil sie reine Textinformationen enthalten, als Übergabeparameter vom Typ *String*[1]*(BStr)* bei einem Methodenaufruf eines COM-Objektes eingesetzt werden.

3.3 Allgemeine Grundlagen

Zu den Allgemeinen Grundlagen, die für das Engineeringsystem genutzt werden, aber auf die hier nicht insbesondere eingegangen werden soll, zählen:

- Relationale Datenbanken.
- Die Datenbanksprache SQL.
- Die Konzepte von Client/Server-Applikationen und Multi-Tier-Applikationen.
- Die Programmiersprache Borland Delphi Version 6.
- Das SPS-Programmiersystem SIMATIC STEP 7.

Für die vorgenannten Themenbereiche sei auf die Standardliteratur verwiesen [19][25][26].

[1] String (engl.): Zeichenkette

4 ANFORDERUNGEN AN DEN DBENGINEER

Bisher wurde der Konstruktionsprozess unter dem Aspekt des Informationsflusses auf der Basis der Datenaustauschdatei MuK-Liste zwischen den einzelnen Fachgebieten betrachtet. Es wurden die unterschiedlichen Sichten (Aspekte) der verschiedenen Beteiligten aufgezeigt. Des Weiteren wurden der Aufbau und die Funktionsweise der Softwarewerkzeuge, die den Entwicklungsprozess einer Anlage unterstützen, erläutert.

In Kapitel 2.5 wurde gezeigt, dass der typische Konstruktionsprozess dadurch gekennzeichnet ist, dass Teillösungen aus vorhandenen Projekten die Grundlage für ein neues Projekt bilden. Das direkte Kopieren von Projektteilen hat, wie dargestellt, erhebliche Nachteile. Trotz der beschriebenen Nachteile ist für die Projektierung der Grundgedanke auf vorhandene Projektierungslösungen aufzubauen richtig, denn die vorhandenen Lösungen repräsentieren das Projektierungswissen eines Unternehmens. Mit dem **dbEngineer** soll dieses Wissen zentral verwaltet und den am Entwicklungsprozess beteiligten Mitarbeitern des Unternehmens für ihre Arbeit zur Verfügung gestellt werden. Die Verwaltung bedeutet, dass repräsentative Lösungen aus den bestehenden Projekten herausgelöst werden, um als Musterlösungen oder Module für typische Aufgaben in einer Bibliothek aufgenommen zu werden.

Der **dbEngineer** soll dem Gedanken eines modulorientierten Projektierungsansatzes auf der Basis einer Modulbibliothek folgen. Der Aufbau der Modulbibliothek beruht auf der Zuordnung von elektro-mechanischem Grundsystem zu den darauf abgestimmten Musterlösungen für E-CAD, SPS und HMI-System.

Mit der MuK-Liste verfügt die Automatisierung bereits über die Dekomposition einer Anlage in ihre elektro-mechanischen Grundsysteme und deren Klassifizierung. Der **dbEngineer** kann diesen Grundsystemen für Stromlaufpläne, SPS-Programme und Leitsysteme jeweils ein Modul aus der Modulbibliothek direkt zuweisen. Dabei greift der **dbEngineer** über die MuK-Liste auf die standardisierten Elemente einer Modulbibliothek zurück und kopiert gezielt die benötigten Module aus der Bibliothek in ein neues Projekt, wie in Abbildung 4-1 gezeigt.

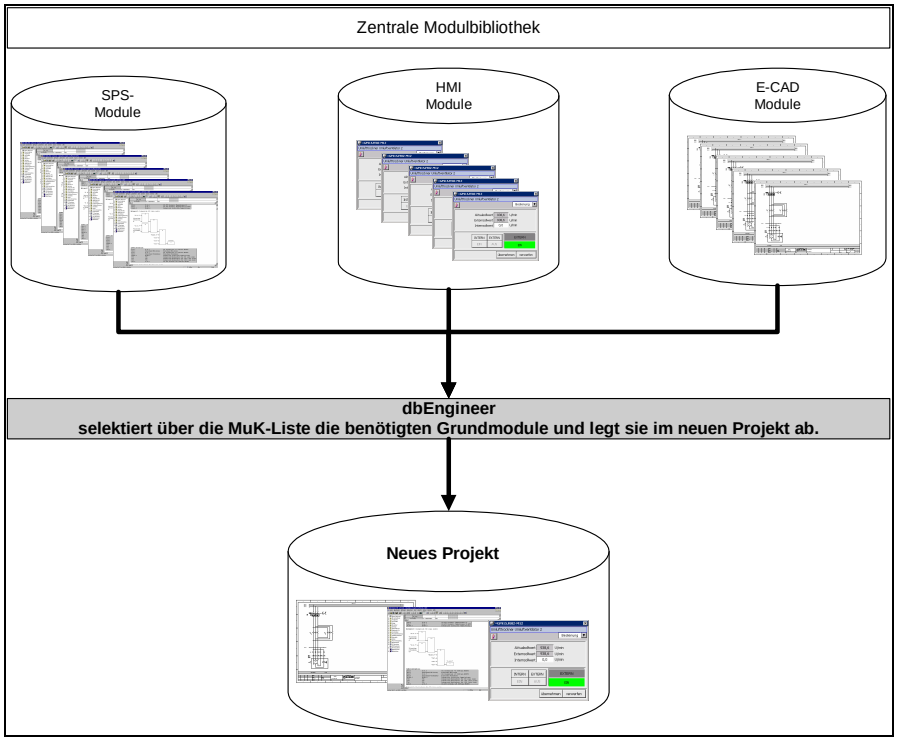

Abbildung 4-1: Modulorientierter Projektierungsansatz mit **dbEngineer**

Mit der Aufnahme von typischen Musterlösungen in eine Bibliothek sollen die daraus entstehenden Module projektneutral abgelegt werden. Das bedeutet, dass die Module nicht mehr wie beim direkten Kopieren mit den technischen Daten oder Referenzkennzeichen der Ursprungsanlage versehen sein dürfen. Die Module sollen anstelle der projektspezifischen Beschriftung mit neutralen Platzhaltervariablen versehen werden. Die neutral gestalteten Module können mit Serienbriefen und deren Feldfunktionen, wie sie aus Textverarbeitungen bekannt sind, verglichen werden. So wie bei der Serienbrieffunktion die Felder mit den konkreten Daten beispielsweise aus einer Adressdatenbank versorgt werden, soll der **dbEngineer** die Platzhalter in den Modulen beim Generierlauf durch Daten der MuK-Liste ersetzen.

Der Aufbau einer Modulbibliothek soll durch das Automatisierungsunternehmen erfolgen, das den **dbEngineer** anwendet. Das Unternehmen kann aus Projektierungshandbüchern oder aus abgewickelten Projekten geprüfte und bewährte Module in die Bibliothek überführen und sie somit für die allgemeine Nutzung im Unternehmen freigeben [27]. Die Modulbibliothek soll das unternehmesweite Projektierungswissen für elektro-mechanische Grundsysteme abbilden.

Im Bereich des hier betrachteten Maschinen- und Anlagenbaus, wie z.B. in der Hütten- und Walzwerktechnik, Petrochemie, Kläranlagen oder Müllverbrennungsanlagen, handelt es sich durchweg um Sondermaschinen, deren Wiederholgrad von Teilanlagen gering ist. Hier überwiegt ganz entscheidend der Anteil der projektbezogenen Sonderkonstruktion, -planung und -fertigung. Trotz der Variationsbereite, die mit der Konstruktion von Sondermaschinen einhergeht, lassen sich diese auf elektro-mechanische Grundelemente zurückführen, wie bei der Vorstellung der MuK-Liste gezeigt wurde. Der **dbEngineer** soll daher im Bereich des Basisengineerings den Konstruktionsprozess, der sich auf Grundelemente von elektro-mechanischen Systemen zurückführen lässt, unterstützen und verkürzen.

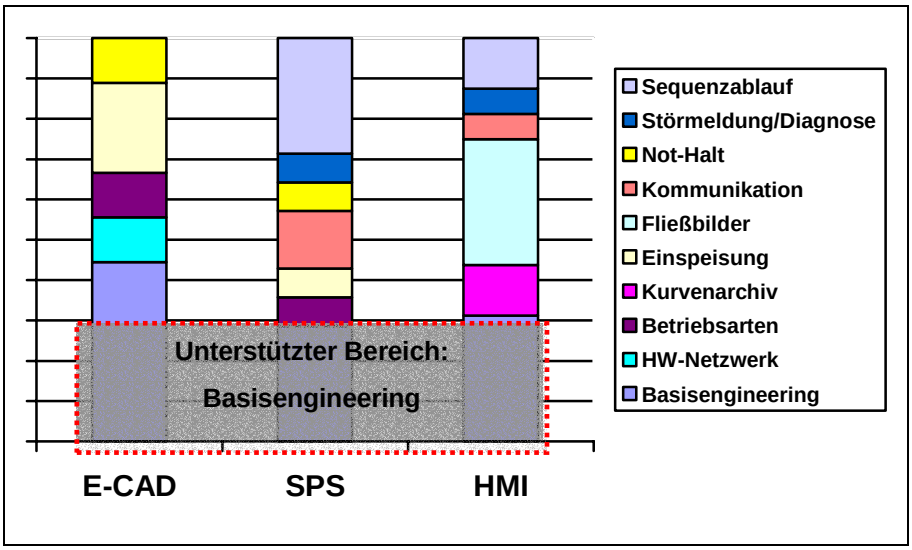

Abbildung 4-2: Arbeitspakete in den Fachgebieten und vom **dbEngineer** unterstützter Bereich

Abbildung 4-2 zeigt qualitativ die verschiedenen Projektierungsleistungen in der Automatisierungstechnik und den Bereich des Basisengineerings, der vom **dbEngineer** unterstützt werden soll. Bei den anderen Tätigkeiten handelt es sich durchweg um projektspezifische Sonderaufgaben, die im Rahmen der Pflichtenhefterstellung spezifiziert werden. Mit den projektspezifischen Sonderlösungen, beispielsweise in den Bereichen Sequenzablauf, Kommunikation oder Störmeldung/Diagnose, ist häufig ein Alleinstellungsmerkmal gegenüber Wettbewerbern verbunden [1]. Es ist deshalb das Ziel der Automatisierungslieferanten, für den Teil der individuellen Sonderplanung die Planungskapazität zu erhöhen, was mit der Forderung einher geht, die Aufwendungen für Grundkonstruktionen zu verringern, um die Gesamtkosten für das Engineering konstant oder geringer zu halten. Mit einer Vereinfachung des Basisengineerings greift der **dbEngineer** genau diesen Gedanken auf.

Um diesen Aspekt der Engineeringunterstützung zu verdeutlichen, soll der qualitative Terminplan, wie er in Kapitel 2.4 *Planung in der Automatisierung* vorgestellt wurde, einem ebenfalls qualitativ aufgetragenen Zeitablauf, wie er mit dem Einsatz des **dbEngineers** entsteht, in Abbildung 4-3 gegenübergestellt werden.

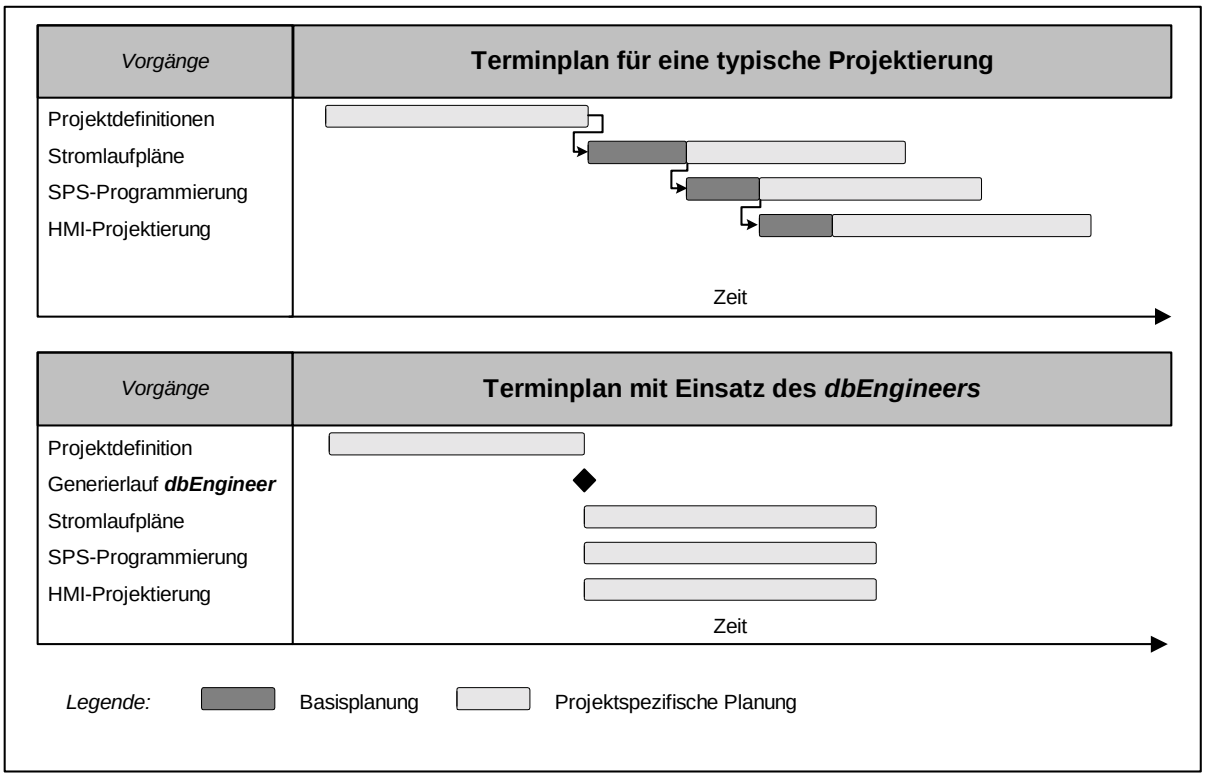

Abbildung 4-3: Vergleich: Typische Projektierung und gleichzeitiger Planungsbeginn bei Einsatz des **dbEngineers**

Der **dbEngineer** soll den Bereich der Basisplanung verkürzen, wodurch die einzelnen Fachgebiete sofort und gleichzeitig mit ihrer Detailplanung beginnen können, weil die benötigten Informationen für alle Beteiligten mit dem Generierlauf erzeugt wurden. Gegenüber der typischen Projektierung mit gestaffelten und abhängigen Startterminen, soll der mit **dbEngineer** unterstützte Ablauf den Gesamtzeitraum, der für Stromlaufpläne, SPS- und HMI-Projektierung benötigt wird, deutlich verkürzen. Die Durchführung einer Projektdefinition ist für beide Abläufe gleichermaßen erforderlich und kann wegen ihrer projektbezogenen Ausprägung nicht automatisiert werden.

Der **dbEngineer** soll die Integration der verschiedenen Werkzeuge in der Automatisierung auf zwei Wegen unterstützen:

1. Durch Kopplungsmöglichkeiten zu den verschiedenen Software-Werkzeugen.

2. Durch Bereitstellung einer Modulbibliothek mit Modulen für alle an den **dbEngineer** angebundenen Werkzeuge.

Von existentieller Bedeutung für die Referenzierung von Modulen der Bibliothek ist die automatische Auswertung der Datenaustauschliste (MuK-Liste), die für diese Verwendung neu zu definieren ist.

4.1 Anforderungen an ein Datenaustauschformat

Das Datenaustauschformat spielt, wie gezeigt, im Planungsprozess eine zentrale Rolle, daher sind die Anforderungen so zu stellen, dass alle Beteiligten am Planungsprozess zu wirklichen Vorteilen bei der Anwendung kommen.

4.1.1 Allgemeine Anforderugnen

Die Allgemeinen Anforderungen betreffen nicht die Spezifikation und Definition eines Austauschformates, sondern vielmehr die Rahmenbedingungen, die für Entwicklung eines Austauschformates einzuhalten sind:

Textbasiert

Das Datenformat darf kein spezielles oder proprietäres Format definieren. Das Format soll nur Tetxinformationen enthalten und Sonderzeichen wie Umlaute unterstützen, um international eingesetzt werden zu können.

Plattformübergreifend

Das zu erstellende Format darf nicht an eine Betriebssystemplattform gebunden sein. Gleichzeitig soll die Möglichkeit bestehen, das Dokument auf einem Datenträger abzulegen, als E-Mail-Anhang oder über das Internet auszutauschen.

Verarbeitung

Der Aufbau und die Struktur des Formates sind so zu wählen, dass eine Verarbeitung in EDV-Systemen möglich ist, gleichzeitig aber das Dokument von einem Menschen gelesen und nachvollzogen werden kann.

4.1.2 Anforderungen als Format für eine MuK-Liste

Für die Verwendung als MuK-Liste und deren Nutzung im Projektierungsablauf sind zusätzliche Forderungen zu stellen:

Eindeutige Kennzeichnung von Aktoren und Sensoren

Für jeden Aktor oder Sensor ist ein eindeutiges Kennzeichen zu definieren, mit dem das Bauteil während des gesamten Planungsprozesses identifiziert werden kann.

Abbildung der Beziehungen zwischen Aktor und Sensor

Um die automatische Auswertung der MuK-Liste in EDV-Systemen zu ermöglichen, muss das Datenaustauschformat die Abhängigkeit zwischen Aktor und Sensoren als elektro-mechanischem Grundsystem eindeutig abbilden können.

Unterstützung der Aspekte nach DIN EN 61346

Um ihrer Aufgabe als Kommunikationsmittel zwischen Investor, Anlagenbau und Automatisierung nachkommen zu können, soll einer Komponente direkt im Datenformat die verschiedenen Referenzkennzeichen der Aspekte zuordnet werden können.

Nutzung der bekannten Abkürzungen

Die Akzeptanz bei den Nutzern stellt ein wichtiges Kriterium für die Verbreitung eines Daten- und Austauschformates dar. Daher soll das zu entwickelnde Datenformat die bekannten und seit Jahren bewährten Abkürzungen und Bedeutungen weitestgehend unterstützen. Zugunsten der Akzeptanz soll auf eine Neudefinition verzichtet werden.

4.2 Anforderungen an den dbEngineer in der Automatisierung

Nachdem die Spezifiktionen für das Datenaustauschformat vorliegen, soll ein Anforderungskatalog formuliert werden, der bei der Entwicklung des **dbEngineers** einzuhalten ist. Der Katalog ist so aufzustellen, dass der Anwender durch Nutzung des **dbEngineers** zu tatsächlichen Vorteilen kommt. Ausgehend von Allgemeinen Anforderungen, der Unterstützung im Engineeringprozess, über die Datenbankseite, den Kopplungsmöglichkeiten zu Entwicklungs- und Fremdsystemen, bis

hin zum Einsatz in verteilten Systemen sollen im Folgenden die Anforderungen an den **dbEngineer** spezifiziert werden.

4.2.1 Allgemeine Anforderungen

Bei den Allgemeinen Anforderungen geht es weniger um die zu realisierende Funktion des **dbEngineers** als um die allgemeinen Forderungen, die an ein Softwaresystem zu stellen sind. Dabei handelt es sich um:

Modularen Aufbau

Der **dbEngineer** muss modular aufgebaut sein, wobei zusammengehörige gleichartige Funktionen ein gemeinsames Modul bilden. Der **dbEngineer** lässt sich damit für unterschiedliche Einsatzzwecke konfigurieren.

Erweiterbarkeit durch Softwaremodule

Der Erweiterbarkeit des **dbEngineers** kommt bei den Anforderungen besondere Bedeutung zu. Der Anwender oder Dritte als Dienstleister müssen die Möglichkeit haben, das System mit neuen Softwaremodulen nach ihren Vorstellungen zu erweitern und anzupassen.

Schnittstelle für neue Entwicklungssysteme

Die Schnittstelle zu einem Entwicklungssystem muss als eigenständiges Softwaremodul ausgeführt werden. Durch Installation neuer Module oder Austausch vorhandener Module muss der Anwender die Möglichkeit haben, den **dbEngineer** um den Zugriff auf neue Planungssysteme oder auf aktualisierte Versionen bestehender Planungssysteme zu erweitern. Der **dbEngineer** definiert eine eindeutige Schnittstelle für die Integration der Module. Alle Module, die mit dem **dbEngineer** zusammenarbeiten wollen, müssen sich an die Konventionen der Schnittstelle halten.

Installation ausschließlich als Binärcode

Der **dbEngineer** wird ausschließlich in kompilierter ausführbarer Form ausgeliefert. Anwender und Drittanbieter von Softwaremodulen dürfen keine Annahmen über Arbeitsweisen und Implementierungen der Software des **dbEngineers** machen. Für die Anbindung und Erweiterung durch Dritte dürfen ausschließlich die dokumentierten Schnittstellen des **dbEngineers** benutzt werden.

Programmiersprachenunabhängige Schnittstellen

Damit die Akzeptanz des **dbEngineers** bei möglichst vielen Anwendern gegeben ist, müssen die Schnittstellen des Systemkerns offen gelegt und dokumentiert werden. Die Schnittstellen selber dürfen nicht den Einsatz einer bestimmten Programmiersprache fordern, sondern müssen sprachunabhängig sein, damit jeder Anwender mit seiner Programmiersprache Erweiterungen vornehmen kann, was wiederum die Akzeptanz des Gesamtsystems erhöht.

4.2.2 Anforderungen an die Engineeringunterstützung

Die Engineeringunterstützung ist der zentrale Aufgabenbereich des **dbEngineers**. Von der Engineeringunterstützung hängen die Verwendungsmöglichkeiten des gesamten Systems ab. Zu den typischen Leistungen der Unterstützung gehören:

Generieren des Grundengineerings

Das Generieren eines Grundengineerings für typische elektro-mechanische Grundsysteme ist die Kernaufgabe des **dbEngineers**. Da sich die Generierungsläufe für die einzelnen Gebiete E-CAD, SPS und Leitsysteme von ihren Anforderungen deutlich unterscheiden, werden die Spezifikationen einzeln aufgeführt.

Anforderungen für E-CAD

- Ermittlung des zugehörigen Zeichnungsmakros (Modul).
- Übernahme des Funktions- und Ortsaspekts.
- Versorgung der Platzhaltervariablen im Modul.
- Generieren des Stromlaufplans aus dem Modul.

Anforderungen für die SPS

- Ermittlung des zugehörigen Programmmoduls.
- Versorgung der Platzhaltervariablen im Modul.
- Symboltabellen für Variable anlegen und beschreiben.
- Generieren des Programmcodes.

Anforderungen für Leitsysteme

- Ermittlung des zugehörigen Pop-Up-Moduls.
- Versorgung der Platzhaltervariablen im Pop-Up-Modul.
- Symboltabellen für Variable anlegen und beschreiben.
- Anbindung an die Variablen der SPS vornehmen.

Import und Export einer MuK-Liste

Der **dbEngineer** muss in der Lage sein, eine MuK-Liste in einem geeigneten allgemein anerkannten Datenformat zu importieren und zu exportieren.

Unterstützung der Aspekte nach DIN EN 61346-1

Der **dbEngineer** soll die drei Aspekte der DIN EN 61346-1 anzeigen und verwalten können:

- Produktionsaspekt
- Funktionsaspekt
- Ortsaspekt.

Durch die parallele Nutzung und Unterstützung aller drei Aspekte hat jedes Fachgebiet entsprechend der festgelegten Bezeichnungskreise die Möglichkeit auf die Komponenten zuzugreifen. Die Fehlerquelle der unterschiedlichen Bezeichnungen in Anlagenbau und Automatisierung wird damit ausgeräumt.

Verwalten von Modulbibliotheken

Der **dbEngineer** soll die Verwaltung der unternehmensweit gültigen Modulbibliothek übernehmen. Die Modulbibliothek ist so zu organisieren, dass der lesende Zugriff auf Module der Datenbank für alle Projektmitarbeiter des Unternehmens zugelassen ist. Das Verändern der Modulbibliothek soll nur für Administratoren erlaubt sein, um ein zufälliges oder unbeabsichtigtes Modifizieren der Modulbibliothek zu verhindern. Gleichzeitig soll damit das Freigeben neuer oder revisionierter Module und der Aufbau einer Revisionshistorie unterstützt werden.

Zur Administration der Modulbibliothek gehören:

- Hinzufügen neuerstellter freigegebener Module
- Archivieren veralteter Module (Revisionshistorie)
- Aufnahme von revisionierten freigegebenen Modulen.

Die vorgenannten Tätigkeiten sind nur für den Systemadministrator zugelassen.

Verwalten von Projekten

Typischerweise werden in einem Unternehmen mehrere Projekte gleichzeitig abgewickelt. Der **dbEngineer** muss den Zugriff auf verschiedene Projekte zulassen.

Verwalten von Globalen Projektdefinitionen

In der Globalen Projektdefinition werden die notwendigen Daten, Parameter und Einstellungen für die Arbeit im Projekt hinterlegt. Für die angebundenen Werkzeuge, wie Softwareadapter zum Generieren des Engineerings, sind die projektgültigen Regeln für Zugriffe und Generierung eingetragen.

Typische Informationen der Globalen Projektdefinition sind z.B.:

- SPS-Type und zughörige Programmiersoftware
- Projektname in der SPS-Programmierung
- SPS: Rechner IP-Adresse (für Netzwerk-Zugriffe)
- Zulässiger Bereich bei der automatischen Adressvergabe in der SPS.

Die Einstellungen sind nicht nur für die exemplarisch aufgeführten Einstellwerte der SPS vorhanden, sondern für alle Werkzeuge sind entsprechende Rubriken vorhanden.

Zugriffswege und Projektnamen, die in den einzelnen Softwarewerkzeugen hinterlegt werden, sind für die Zusammenarbeit im Verbund wichtig, weil sich am Projekt beteiligte Rechner an unterschiedlichen Standorten befinden und dort gleichzeitig für unterschiedliche Projekte aktiv sein können. Es muss sichergestellt werden, dass sich ein Softwareadapter ausschließlich mit dem gewünschten Rechner und dort nur mit dem ausgewählten Zielprojekt verbindet.

4.2.3 Anforderungen an die Datenbankseite

Der **dbEngineer** fordert von der einzusetzenden Datenbank die Erfüllung zweier ISO Standards:

- ISO 9579:2002 [28] - Netzweiter Datenbankzugriff
- ISO 9075:1989 [29] – SQL-2-Standard

Die beiden vorgenannten Standards werden von allen wichtigen Datenbankensystemen unterstützt. Eine Nutzung der Funktionen der ISO-Standards stellt demnach kein Hindernis für die Portierbarkeit auf andere DBMS dar.

Die Anforderungen des **dbEngineers** an die Unabhängigkeit vom eingesetzten DBMS beschränken sich auf:

Unabhängigkeit von einem DMBS

Der **dbEngineer** soll bei möglichst vielen Anwendern eingesetzt werden können, die möglicherweise bereits für ihre Geschäftsprozesse ein DBMS benutzen. Aus wirtschaftlichen Gründen wird der Anwender fordern, dass auch der **dbEngineer** dieses DBMS für die Ablage seiner Daten nutzt. Die Schnittstelle des **dbEngineers** muss so konfigurierbar sein, dass sie auf unterschiedliche DBMS zugreifen kann, ohne dass ein Neukompilieren erforderlich ist.

Unabhängigkeit der Inhalte vom DBMS

Trotz der Möglichkeit unterschiedliche DBMS anbinden zu können, ist ein System noch nicht vollständig unabhängig vom eingesetzten DB-System. Erst durch den Verzicht auf den Einsatz DB-systemspezifischer Eigenschaften wie die Deklaration nativer, proprietärer Datentypen, Nutzung von DB-Triggern und Stored Procedures[1] wird der **dbEngineer** vollständig unabhängig vom eingesetzten Datenbanksystem. Der **dbEngineer** darf sich nur auf abstrakte, für alle Datenbank-Systeme gültige, Aufrufe und Funktionalitäten abstützen.

[1] Stored Procedures (engl.): gespeicherte Prozeduren (hier: im DB-System selber hinterlegte Prozeduren)

4.2.4 Anforderungen an die Kopplungsmöglichkeiten

Die Kopplung zu den Engineeringwerkzeugen ist die zentrale Funktion des **dbEngineers** für den Einsatz in der Projektierung. Wie Abbildung 4-4 darstellt, soll der **dbEngineer** modular aufgebaut und Fremdsysteme über spezielle Softwareadapter an das Kernsystem angebunden werden.

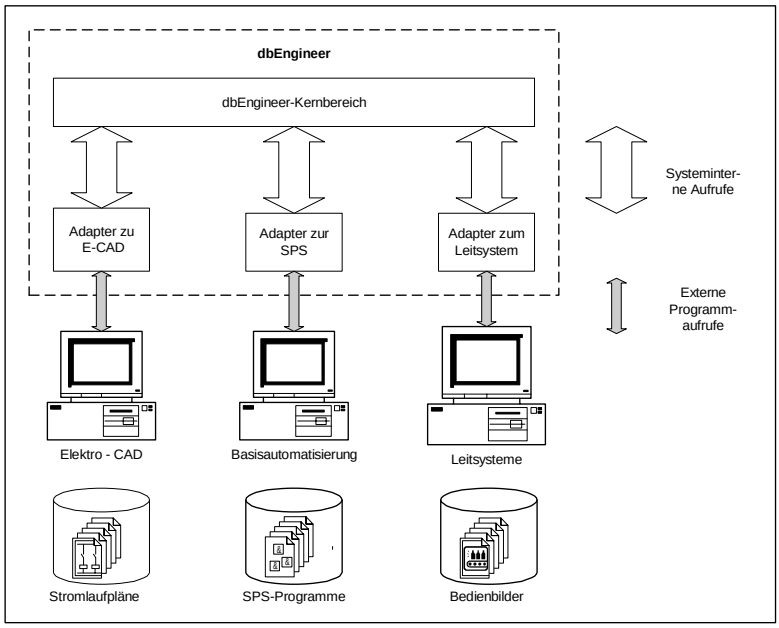

Abbildung 4-4: Kopplung des dbEngineers zu den Engineeringwerkzeugen

Der **dbEngineer**-Kern darf keinerlei Annahmen über die Funktionalität und interne Wirkungsweise der angekoppelten Engineeringwerkzeuge machen, dies ist ausschließlich Sache der speziellen Softwareadapter. Wie in Kapitel 4.2.1 Abschnitt *Schnittstelle für neue Entwicklungssysteme* erläutert, sind die Adapter zwar Teil des **dbEngineers**, können aber durch Dritte erstellt und im **dbEngineer** integriert werden, wenn sie sich an die Spezifikationen der veröffentlichten internen Schnittstelle halten. Die Schnittstelle zwischen Softwareadapter und Engineeringsystem wird vom Hersteller des Engineeringwerkzeugs veröffentlicht und beschrieben.

Die anzubindenden Projektierungssysteme bieten unterschiedlichste Schnittstellen für den Zugriff durch Fremdsysteme an, wie in den Tabelle 4-1 bis Tabelle 4-3 für einige typische Engineeringwerkzeuge aufgelistet [8][33]. Die eingesetzten Softwareadapter zu den Anwendungen müssen diesem Umstand Rechnung tragen und die Schnittstelle gegenüber dem **dbEngineer** derart abstrahieren, dass die Zugriffe für den **dbEngineer** transparent sind.

E-CAD	Makrosprache	API-Schnittstelle	Steuerdatei	Blattmakro
Eplan 5.40	Nein	Nein	Ja	Ja

Tabelle 4-1: Zugriffswege auf E-CAD

SPS/-Leitsysteme	Makrosprache	API-Schnittstelle	Steuerdatei	Funktionsvorlagen
ABB AC800F	Nein	Nein	Nein	Nein
Schneider Concept 2.5	Nein	Nein	Nein	Nein
Simatic PCS7 V5.2	Nein	Ja (COM)	Ja (MS-Office)	Ja
Simatic S7 V5.1	Nein	Ja (COM)	Nein	Ja

Tabelle 4-2: Zugriffswege auf SPS-Programmiersysteme und SPS-basierte Leitsysteme

HMI-Systeme/Leitsysteme	Makrosprache	API-Schnittstelle	Steuerdatei	Bildobjekte
ABB AC800F	Nein	Nein	Nein	Ja
Siemens WinCC V5.1	Nein	Ja (API)	Nein	Ja
Simatic PCS7 V5.2	Nein	Ja (API)	Ja	Ja
US Data Factory Link V 7.0	Nein	Ja (COM)	Nein	Ja
Wonderware Intouch V7.1	Nein	Nein	Nein	Ja

Tabelle 4-3: Zugriffswege auf HMI-Systeme und Visualisierung von Leitsystemen

Die oben genannten Systeme stellen im Wesentlichen drei verschiedene Zugriffswege[1] bereit:

- *COM-basiserte Aufrufschnittstelle*: Die Softwareadapter können direkt auf die Anwendung zugreifen.
- *API-basiserte Aufrufschnittstelle*: Die Softwareadapter können direkt auf die Anwendung zugreifen. Die Anwendung muss allerdings auf demselben Rechner installiert sein.
- *Steuerdatei*: Der Softwareadapter generiert eine spezielle Datei. Die Datei wird vom Engineeringwerkzeug importiert und führt dann automatisierte Abläufe durch [30].

Der jeweilige Softwareadapter muss in der Lage sein mit dem Engineeringwerkzeug zu interagieren, ohne dass der **dbEngineer** anpasst werden muss.

[1] **Anmerkung**: In die Tabellen wurde ein Zugriffsweg nur dann aufgenommen, wenn er für den Endnutzer der Software in der Begleitdokumentation des Produktes eindeutig beschrieben und offen gelegt wird. Eventuell bieten die Werkzeuge noch weitere Schnittstellen, die undokumentiert sind. In Sinne der Investitionssicherheit eines Endkunden sind sie aber nicht nutzbar, da sie nicht Teil eines Lizenzvertrages sind und mit jedem Servicerelease, Update und jeder neuen Version der Software ohne Rechtsanspruch geändert oder entfernt werden können.

5 KONZEPT DES DBENGINEERS

Im vorausgehenden Kapitel wurden die Anforderungen an Funktionen, Aufgaben und Einsatzzwecke des **dbEngineers** gemacht. In diesem Kapitel wird ein Konzept vorgestellt, das die vorgenannten Anforderungen erfüllt.

Bevor dieses Konzept entwickelt wird, soll noch einmal verdeutlicht werden, dass der **dbEngineer** Planungs- und Projektierungssysteme integrieren soll, so wie sie zurzeit auf dem Markt angeboten und genutzt werden. Die aktuellen Systeme der Elektrokonstruktion bieten häufig nicht die optimalen Voraussetzungen wie Schnittstellen zur Kopplung oder offen gelegte allgemeine Datenbankkerne zur Speicherung von Daten, wie sie im Bereich der 3D-Konstruktion seit langem bekannt oder als PDM-Kern in der Diskussion sind [2].

Es ist trotzdem sinnvoll die aktuellen Systeme mit in den **dbEngineer** integrieren zu können, weil es sich bei den Systemen zur Anlagenplanung um extrem langlebige Wirtschaftsgüter handelt. Produktionsanlagen haben in der Chemischen Industrie eine auf 35 Jahre abgeschätzte Gesamtanlagenlebensdauer. Die installierten technischen Komponenten werden im Mittel alle 19 Jahre ausgetauscht, die Einrichtungen der Prozessleittechnik haben mit 10 bis 15 Jahren bis zum Austausch eine kürzere Lebensdauer als andere Komponenten. Dementsprechend lange werden auf die Systeme der Anlagenausrüstung von den Herstellern auf dem Markt gehalten: Die SPS-Baureihe SIMATIC S5 und ihr Programmierwerkzeug STEP 5 zum Beispiel wurden im Jahre 1979 vorgestellt und erst für die Jahre 2003 und 2004 in Stufen abgekündigt. In dieser Zeit hat es aus Gründen der Investitionssicherheit bei der Programmiersoftware nur marginale Änderungen gegeben, um die Auf- und Abwärtskompatibilität weitgehend zu erhalten [12][31].

Das System **dbEngineer** ist daher so aufgebaut, dass seine Funktionalität weitgehend skalierbar und modularisiert ist. Dadurch lassen sich heutige und zukünftige Systeme der Anlagenplanung einbinden.

Das gesamte Unterstützungssystem für die Automatisierung im Maschinen- und Anlagenbau, das hier vorgestellt wird, besteht aus zwei Bausteinen:

1. Der Engineeringunterstützung **dbEngineer** als zentraler Komponente
2. Der Datenaustauschdatei MuK-Liste als XML-Dokument.

Zum Gesamtverständnis des **dbEngineers** wird zunächst die grundsätzliche Arbeitsweise der Engineeringunterstützung erläutert. Bei dieser Betrachtung wird die Aufteilung des **dbEngineers** in einzelne Komponenten zu Gunsten der Übersichtlichkeit vernachlässigt.

Das generische und automatische Erzeugen eines Basisengineerings durch den **dbEngineer** basiert auf den Daten, die in der MuK-Liste der Anlage abgelegt sind. Der **dbEngineer** kann auf die Daten der MuK-Liste zugreifen und diese importieren. Durch den Import entfällt die manuelle Eingabe von Daten und Beschreibungen der Anlagenkomponenten. Die importierten Daten werden in einer Projektdatenbank abgelegt und stehen dort für die weitere Bearbeitung zur Verfügung.

Für die projektbezogenen Planungssysteme existiert eine Modulbibliothek mit Musterlösungen für die einzelnen elektro-mechanischen Systeme. Durch die Angaben, die durch den Import der MuK-Liste gewonnen werden, kann der **dbEngineer** die elektro-mechanischen Grundsysteme einer Anlage identifizieren und in der Modulbibliothek die zu ihnen passenden Module bestimmen. Der **dbEngineer** parametriert die so gefundenen Module der Bibliothek mit den zugehörigen technischen Daten und Beschreibungen der Einzelsysteme, wie sie aus der MuK-Liste übernommen oder im **dbEngineer** ergänzt wurden. Der **dbEngineer** trägt die parametrierten Module der elektro-mechanischen Systeme in das jeweilige Planungssystem ein.

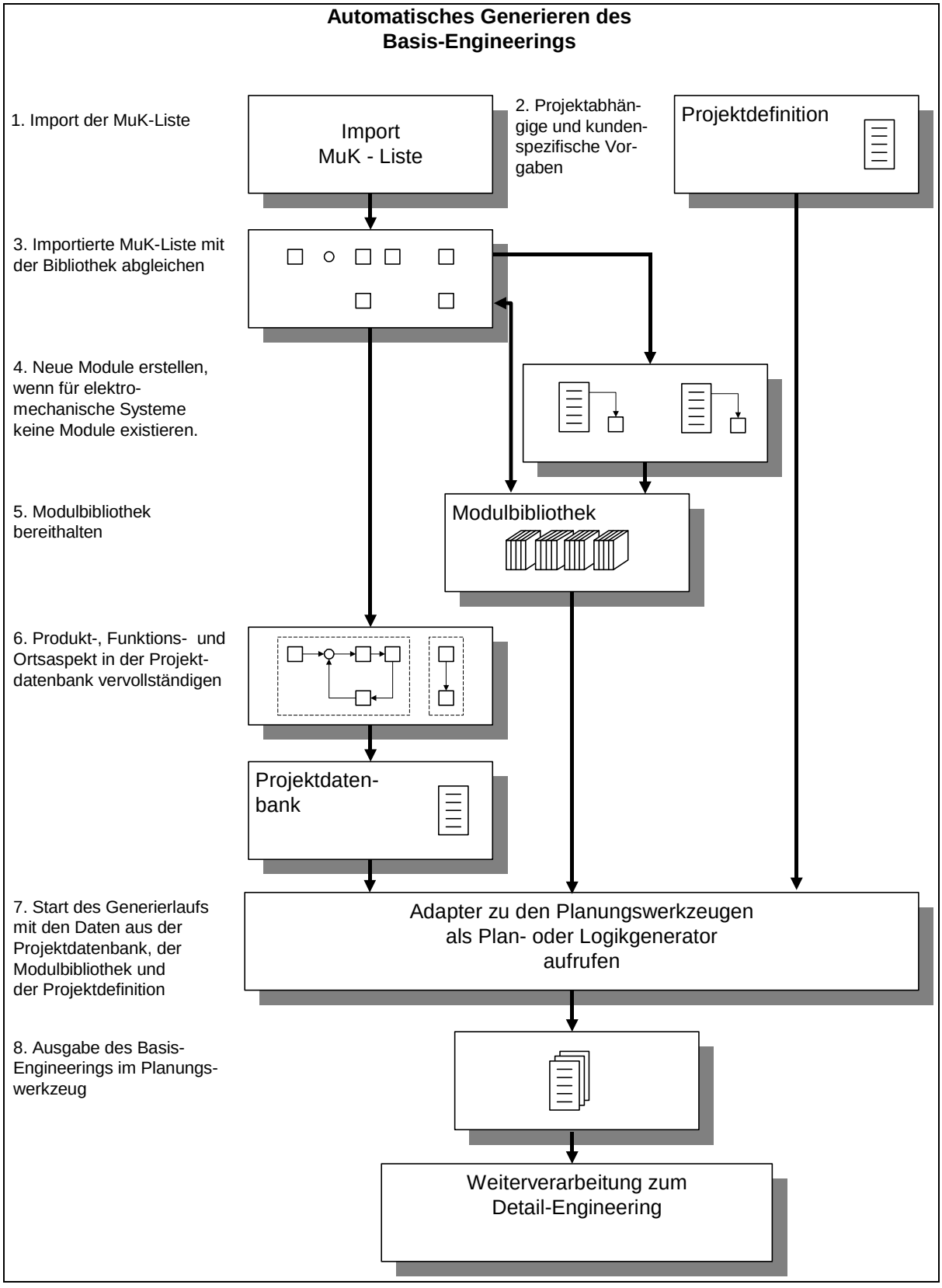

Abbildung 5-1: Prinzipieller Ablauf des automatischen Generierens eines Basisengineerings mit **dbEngineer**

Das Grundprinzip der Erzeugung eines Basisengineerings wird für alle beteiligten Planungswerkzeuge nach demselben Schema gemäß Abbildung 5-1 abgewickelt:

1. Die MuK-Liste des Projektes wird durch den **dbEngineer** importiert. Die Informationen aus der MuK-Liste werden in der Projektdatenbank abgelegt und bilden das informationstechnische Fundament für die weitere Projektierung.
2. Gleichzeitig werden die Projektdefinitionen vom Projekteur erzeugt. Die Projektdefinitionen enthalten allgemeine Festlegungen zum Projekt, wie beispielsweise das CAD-Werkzeug, die SPS oder das einzusetzende Leitsystem.
3. Danach wird geprüft, ob in der Konstellation des Projektes, für die vorgeschriebenen Projektierungswerkzeuge und die in der MuK-Liste spezifizierten elektro-mechanischen Systeme, die benötigten Makros in der Bibliothek vorhanden sind.
4. Falls neue elektro-mechanische Systeme gefunden oder Planungswerkzeuge benutzt werden, für die Module in der Bibliothek fehlen, muss die Bibliothek um diese fehlenden Module ergänzt werden.
5. Die Modulbibliothek ist für den Generiervorgang vervollständigt bereit zuhalten.
6. Es ist zu prüfen, ob für alle Aktoren und Sensoren die Referenzkennzeichen der Aspekte vergeben wurden. Fehlende Kennzeichen müssen durch den Projekteur im **dbEngineer** eingetragen werden.
7. Mit den Daten aus der Projektdatenbank für die elektro-mechanischen Systeme, der Modulbibliothek und den Projektdefinitionen, ruft der **dbEngineer** das jeweilige Planungssystem auf. Die Daten aus der Projektdatenbank dienen jetzt dazu, die Platzhaltervariablen in den Modulen der Modulbibliothek zu ersetzen. Das mit den Realwerten versorgte Modul, wird in das Planungssystem eingetragen.
8. Nach Abschluss des Generierlaufs steht das erzeugte Basisengineering im jeweiligen Planungswerkzeug zur weiteren Bearbeitung durch den Projekteur zur Verfügung. Das Basisengineering wird zum Detailengineering weiterverarbeitet, beispielsweise dadurch, dass die erzeugten Programmteile der SPS-Programmierung in einen übergeordneten Batchablauf eingebunden werden.

Basierend auf den vorgestellten Grundkonzepten wird in den folgenden Kapiteln der **dbEngineer** mit den erforderlichen Komponenten, Datenstrukturen und dem Datenaustauschformat MuK-Liste entwickelt.

5.1 Implementierung des Datenaustauschformates

Die Datenaustauschdatei MuK-Liste stellt die Verbindung zwischen dem Maschinen- und Anlagenbau und der Automatisierungsausrüstung her. Bei der MuK-Liste handelt es sich, wie bereits erläutert, um ein allgemein akzeptiertes und eingesetztes Hilfsmittel zum Informationsaustausch zwischen den beteiligten Fachgebieten. In ihrer jetzigen Form ist die MuK-Liste bereits einige Jahrzehnte alt, sie hat sich jedoch in dieser Zeit als äußerst flexibel und erweiterungsfähig für neue Entwicklungen der Technik, die sie beschreibt, gezeigt. So konnten durch einfaches Definieren von Abkürzungen neue zum Zeitpunkt der MuK-Listenentstehung völlig unbekannte Sensoren (z.B. Ultraschallsensoren) in die MuK-Liste aufgenommen werden.

Bei der Nutzung der MuK-Liste als Datenaustauschformat in Rahmen des **dbEngineers** wird darauf verzichtet, ein völlig neues Format oder System zu entwickeln. Es ist sinnvoller das bekannte Format weiterzuentwickeln, um es in einer modernen Informationsinfrastruktur nutzen zu können. Durch eine gezielte Fortentwicklung wird die Akzeptanz bei den Nutzern vergrößert, weil sie bewährte Strukturen wieder erkennen.

Als Dokumentenformat für den Informationsaustausch wird das XML-Format gewählt. Für das Format spricht, dass es herstellerunabhängig als Standard definiert ist, gleichzeitig aber von einer Vielzahl von Anwendungen, beispielsweise MS-Office XP, Sun StarOffice oder Internet Browser auf unterschiedlichen Betriebssystemplattformen bearbeitet werden kann.

5.1.1 Datenstruktur für die MuK-Liste

Eine Analyse der MuK-Liste zeigt, dass der Aktor das zentrale Objekt der Auflistung ist, mit der Sensorik als Zubehör. Aktor und zugehörige Sensorik bilden und beschreiben dabei ein elektromechanisches Grundsystem, das für die weitere Planung genutzt werden kann. Dargestellt nach den Grundprinzipien der Theorie Relationaler Datenbanken, ergeben diese Grundüberlegungen eine 1:N-Relation zwischen Aktor und Sensor gemäß Abbildung 5-2. Es handelt sich bei hierbei streng genommen nicht um eine Datenbank und deren Modellierung, sondern nur um ein Dokumentenformat zum Datenaustausch.

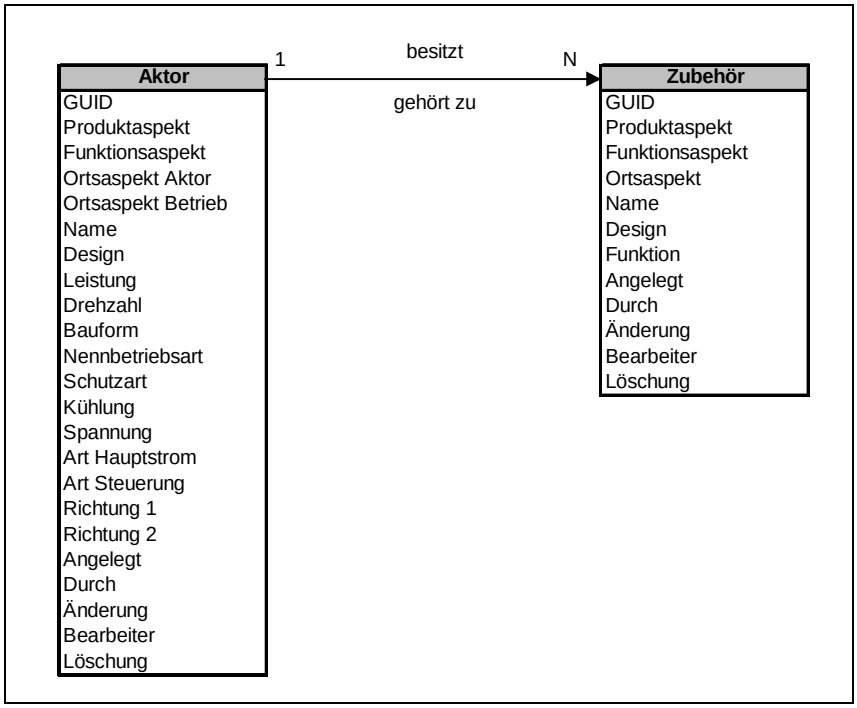

Abbildung 5-2: 1:N-Relation zwischen Aktor und Zubehör in der MuK-Liste

Ausgehend von der Beziehung zwischen Aktor und Sensor und den Feldern der Tabellen (Abbildung 5-2), wird das XML-Format der MuK-Liste gemäß Abbildung 5-3 definiert.

```
<?xml version="1.0" standalone="yes"?>
<DATAPACKET Version="2.0">
<METADATA>
  <FIELDS>
    <FIELD attrname="GUID" fieldtype="string" WIDTH="40">
     <PARAM Name="PROVFLAGS" Value="7" Type="i4" Roundtrip="True"/>
    </FIELD>
    <FIELD attrname="Produktaspekt" fieldtype="string" WIDTH="24"/>
    <FIELD attrname="Funktionsaspekt" fieldtype="string" WIDTH="24"/>
    <FIELD attrname="Ortsaspekt" fieldtype="string" WIDTH="24"/>
    <FIELD attrname="Name" fieldtype="string" WIDTH="40"/>
    <FIELD attrname="Design" fieldtype="string" WIDTH="8"/>
    <FIELD attrname="Leistung" fieldtype="r8"/>
    <FIELD attrname="Drehzahl" fieldtype="r8"/>
    <FIELD attrname="Nennbetriebsart" fieldtype="string" WIDTH="4"/>
    <FIELD attrname="Schutzart" fieldtype="string" WIDTH="4"/>
    <FIELD attrname="Kuehlung" fieldtype="string" WIDTH="8"/>
    <FIELD attrname="Spannung" fieldtype="r8"/>
    <FIELD fieldname="Art Hauptstrom" attrname="Art_Hauptstrom" fieldtype="string" WIDTH="8"/>
    <FIELD fieldname="Art Steuerung" attrname="Art_Steuerung" fieldtype="string" WIDTH="8"/>
    <FIELD fieldname="Richtung 1" attrname="Richtung_1" fieldtype="string" WIDTH="40"/>
    <FIELD fieldname="Richtung 2" attrname="Richtung_2" fieldtype="string" WIDTH="40"/>
    <FIELD attrname="Angelegt" fieldtype="date"/>
    <FIELD attrname="Durch" fieldtype="string" WIDTH="25"/>
    <FIELD attrname="Aenderung" fieldtype="date"/>
    <FIELD attrname="Bearbeiter" fieldtype="string" WIDTH="25"/>
    <FIELD attrname="Loeschung" fieldtype="boolean"/>
    <FIELD attrname="wwTable2" fieldtype="nested">
    <FIELDS>
      <FIELD attrname="GUID" fieldtype="string" WIDTH="40">
      <PARAM Name="PROVFLAGS" Value="7" Type="i4" Roundtrip="True"/>
      </FIELD>
```

```
      <FIELD attrname="Produktaspekt" fieldtype="string" WIDTH="24"/>
      <FIELD attrname="Funktionsaspekt" fieldtype="string" WIDTH="24"/>
      <FIELD attrname="Ortsaspekt" fieldtype="string" WIDTH="24"/>
      <FIELD attrname="Name" fieldtype="string" WIDTH="40"/>
      <FIELD attrname="Design" fieldtype="string" WIDTH="8"/>
      <FIELD attrname="Funktion" fieldtype="string" WIDTH="40"/>
      <FIELD attrname="Angelegt" fieldtype="date"/>
      <FIELD attrname="Durch" fieldtype="string" WIDTH="25"/>
      <FIELD attrname="Aenderung" fieldtype="date"/>
      <FIELD attrname="Bearbeiter" fieldtype="string" WIDTH="25"/>
      <FIELD attrname="Loeschung" fieldtype="boolean"/>
      </FIELDS>
    <PARAMS DEFAULT_ORDER="16396" PRIMARY_KEY="1" LCID="1033"/>
  </FIELD>
</FIELDS>
<PARAMS MD_FIELDLINKS="21 1 12" DEFAULT_ORDER="1" PRIMARY_KEY="1" LCID="1033"/>
</METADATA>
<ROWDATA>
  <ROW GUID="" Produktaspekt="" Funktionsaspekt="" Ortsaspekt="" Name="" Design="" Leistung=""
       Drehzahl="" Nennbetriebsart="" Schutzart="" Kuehlung="" Spannung="" Art_Hauptstrom=""
       Art_Steuerung="" Richtung_1="" Richtung_2="" Angelegt="" Durch="" Aenderung=""
       Bearbeiter="" Loeschung="">
  <wwTable2>
    <ROWwwTable2 GUID="" Produktaspekt="" Funktionsaspekt="" Ortsaspekt="" Name="" Design=""
                 Funktion="" Angelegt="" Durch="" Aenderung="" Bearbeiter="" Loeschung=""/>
  </wwTable2>
  </ROW>
</ROWDATA>
</DATAPACKET>
```

Abbildung 5-3: Definition des XML-Formats für die MuK-Liste

XML ist in der Lage durch Schachtelung von Tags eine solche Zusammengehörigkeit von Aktor und Sensorik abzubilden. Die Mechanismen und Syntax für eine derartige Strukturierung wurden im Kapitel 3.2 *Extensible Markup Language (XML)* diskutiert. Die Abbildung 5-4 zeigt eine MuK-Liste im XML-Dokumentenformat nach ihrem Aufruf in einem Webbrowser.

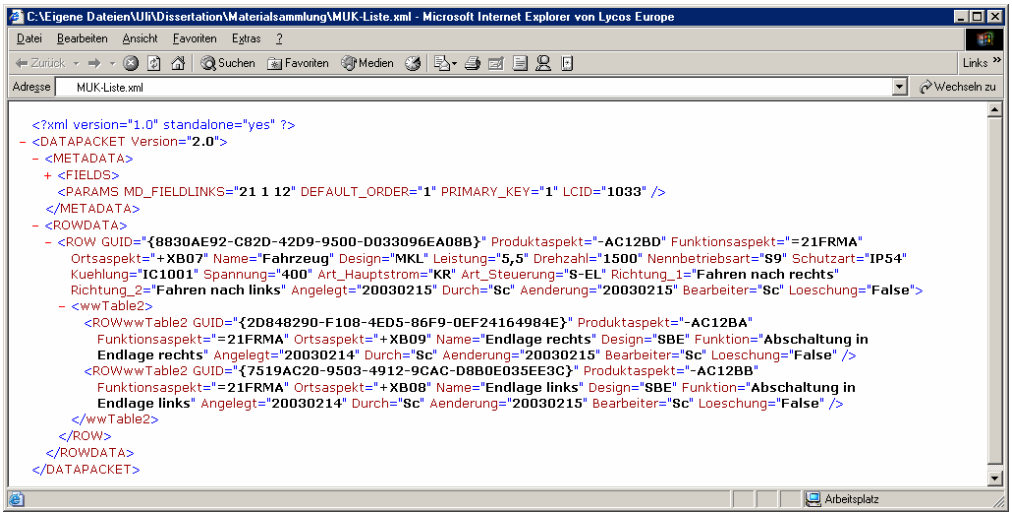

Abbildung 5-4: XML-Datei einer MuK-Liste dargestellt in einem Webbrowser

Für eine spätere automatische Auswertung und einem Versionshandling wurde die MuK-Liste um folgende Datenfelder erweitert:

- *GUID*[1]: Dient der systemweiten eindeutigen Identifikation des Aktors oder Sensors. Wird einmalig beim Anlegen erzeugt.
- *Produktaspekt*: Zeigt die Einordnung innerhalb des Schemas Produktaspekt.
- *Funktionsaspekt*: Zeigt die Einordnung innerhalb des Schemas Funktionsaspekt.
- *Ortsaspekt Aktor*: Zeigt die Einordnung innerhalb des Schemas Ortsaspekt, für den Aktor.
- *Ortsaspekt Betrieb*: Zeigt die Einordnung innerhalb des Schemas Ortsaspekt, für die Schaltgeräte des Aktors.
- *Angelegt*: Datum der Erzeugung des Aktors in der Datenstruktur. Wird einmalig beim Anlegen erzeugt.
- *Durch*: Dokumentiert, wer den Aktor in der Datenstruktur aufgenommen hat. Wird einmalig beim Anlegen erzeugt.
- *Änderung*: Datum der letzten Änderung. Wird bei Änderungen jedes Mal neu eingetragen.
- *Bearbeiter*: Dokumentiert, wer die letzte Änderung vorgenommen hat. Wird bei Änderungen jedes Mal überschrieben.
- *Löschung*: Gelöschte Daten werden in dem Dokumentenformat nur als gelöscht markiert, aber weiterhin in der MUK-Liste mitgeführt.

Die übergebenen Informationen entsprechen ansonsten der bereits bekannten Spalteninformation der MuK-Liste, wie sie derzeit eingesetzt wird. Die Aufgabe der neu definierten Felder *GUID* und *Löschung* ist von besonderer Wichtigkeit für das Gesamtsystem **dbEngineer** und wird deshalb in eigenen Kapiteln diskutiert.

5.1.2 Identifikation von Aktoren/Sensoren

Die MuK-Liste ist ein Informationsmittel, das während der gesamten Anlagenentwicklung von allen Beteiligten zum Datenaustausch genutzt wird. Während der Entstehungsphase einer Anlage muss wegen der Aufteilung von Aufgaben auf mehrere Abteilungen oder Unternehmen erwartet

[1] GUID Abk. für **G**lobally **U**nique **Id**entifier (engl.): Weltweit eindeutiger Bezeichner.

werden, dass dort parallel und gleichzeitig an dem Projekt gearbeitet wird, wodurch die MuK-Liste laufenden Änderungen unterworfen ist.

Es muss deswegen ein Kennzeichen gefunden werden, dass einen Aktor oder Sensor über seinen gesamten Nutzungszyklus hinweg, eindeutig identifiziert. Die Aspekte nach DIN EN 61 346 oder die Zuordnung eines Aktors/Sensors zu den Aspekten können dazu nicht herangezogen werden, weil sie während der Projektierung Änderungen unterliegen können. Eine einfache Zählziffer reicht ebenso nicht aus, weil sie doppelt vergeben werden könnte, wenn im Anlagenbau und Automatisierung unabhängig voneinander Komponenten zugefügt werden. Wird dann die MuK-Liste ausgetauscht, kommt es zu Duplikaten von Identifizieren. Zur Lösung der Konflikte könnte jedem Beteiligten ein Bereich zugewiesen werden, aus dem seine Zählnummern kommen dürfen. Dem steht jedoch ein hoher Verwaltungsaufwand entgegen: Es muss festgestellt werden, welche die letzte benutzte Zählnummer ist, um eine Nächste zu erzeugen oder es kommen neue Firmen zum Projekt hinzu, denen ein weiterer Nummernkreis zugeteilt werden muss.

Zur Identifizierung wird deshalb auf einen GUID zurückgegriffen. Bei einem GUID handelt es sich um eine 128-Bit große Zahl, für die garantiert wird, dass sie weltweit nur ein einziges Mal existiert. Die meisten Betriebssysteme oder Programmiersysteme bieten die Möglichkeit eine solche eindeutige Zahl zu generieren.

Ein GUID wird aber nicht als Zahl sondern als Textstring im Format

{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}

dargestellt, wobei X für eine hexadezimale Zahl (0-9 oder A-F) steht [32].

Durch Einführung des GUID in die MuK-Liste kann jeder der Beteiligten für neu angelegte Komponenten selber GUIDs erzeugen und damit identifizieren, ohne dass es zu Duplikaten kommen wird. Eine Verwaltung zur GUID-Vergabe zwischen den Beteiligten ist nicht erforderlich. Siehe hierzu auch Abbildung 5-4 *XML-Datei einer MuK-Liste dargestellt in einem Webbrowser* Seite 70.

```
procedure TMuKListe.ADOAktorAfterInsert(DataSet: TDataSet);
begin
  With DataSet do
  begin
    FieldByName('GUID').AsString := CreateClassID;
  end;
end;
```

Abbildung 5-5: Zuweisung eines GUIDs als String an das Datenfeld GUID (Borland Delphi)

Abbildung 5-5 zeigt die Zuweisung eines GUID-Strings an das Feld *GUID* der Datenmenge *DataSet* mit dem Programmiersystem Borland Delphi 6. Der String selber wird von der Funktion *CreateClassID* zurückgeliefert.

Für die Verwendung von GUIDs im Zusammenhang mit der Austauschliste gelten folgende Vorschriften:

1. Für jedes neu eingefügte Objekt wird sofort ein neuer GUID als String generiert.
2. Existierende GUIDs, zum Beispiel von verworfenen Objekten, dürfen nicht weiter verwendet werden.
3. Ist einem Objekt ein GUID zugewiesen, darf jeder andere Prozess nur lesend auf den GUID zugreifen. Änderungen, Neugenerieren und Löschen des GUIDs an diesem Objekt sind untersagt.
4. Beim Erzeugen einer MuK-Liste als Datei für den Export ist zu prüfen, ob für jedes Objekt in der MuK-Liste ein GUID existiert. Eine MuK-Liste darf kein Objekt ohne GUID enthalten.
5. Importiert ein Programm eine MuK-Liste, so wird die Identifikation der Objekte über die GUIDs abgewickelt. Existiert im Datenbestand bereits ein Objekt mit diesem GUID, so kann nach Maßgabe der Programmeinstellung ein Überschreiben oder eine andere Reaktion erfolgen. Existiert der GUID noch nicht im Projekt, wird das Objekt vom Programm neu angelegt.

5.1.3 Gelöschte Komponenten

Im Anlagenbau ist es üblich, Teile von Anlagen als Option zu bestellen und während der Projektabwicklung zu entscheiden, ob Optionen realisiert werden oder nicht. Werden im Zuge der Anlagenprojektierung elektro-mechanische Komponenten oder Zubehörteile nicht benötigt, weil sie durch Änderungen der Planung entfallen, müssen sie in der MuK-Liste und den Aspektesichten für die Planungswerkzeuge gekennzeichnet werden.

Entfallene Komponenten verbleiben im Datenbestand des Projektes, sie werden nicht physisch gelöscht, sondern der Datensatz wird als gelöscht markiert. Diese Komponenten werden dem Anwender in der Aspektesicht des **dbEngineers** nicht mehr angezeigt, sie können nicht weiterbearbeitet werden und sind von den Generiervorgängen des **dbEngineers** ausgenommen.

Durch den Verbleib nicht mehr benötigter Komponenten im System werden mehrere Vorteile erzielt. Das Updaten der Projektdaten durch den Import der MuK-Liste wird stringenter. Würden die Komponenten physisch aus dem Datenbestand gelöscht, müsste beim Import einer MuK-Liste indirekt aus dem Fehlen einer Komponente abgeleitet werden, dass sie gelöscht wurde. Mit dem Verbleib einer gelöschten Komponente im System wird der Aufbau einer Revisionshistorie möglich.

Die Kennzeichnung einer gelöschten Komponente erfolgt über das Feld *Löschung*, wobei folgende Bedeutung festgelegt ist:

- *False*[1] *oder Leer:* Die Komponenten ist in der aktuellen Projektierung der Anlage aktiv und wird genutzt.
- *True*[2]*:* Die Komponente ist gelöscht und wird im Projekt nicht mehr verwendet.

An Hand der Löschinformation führt ein Programm, das die MuK-Liste importiert, den Status *Löschung* in seiner Datenbank nach.

Für den Einsatz des Merkmals *Löschung* gelten unter Berücksichtigung der Zusammenhänge gemäß des ER-Diagramms Abbildung 5-2 auf Seite 69, folgende Regeln:

1. Erhält eine Komponente das Kennzeichen *Gelöscht*, so muss zwangsweise das zugeordnete Zubehör ebenfalls das Kennzeichen *Gelöscht* erhalten, weil das Zubehör als relational abhängig nicht allein stehen darf.
2. Das Zubehör darf einzeln und selektiv die Kennung *Gelöscht* erhalten.

5.2 Das Engineeringtool dbEngineer

Bei der Architektur des Engineeringtools **dbEngineer** kommt der Forderung nach Skalierbarkeit und modularem Aufbau eine besondere Bedeutung zu. Wegen der überragenden Bedeutung für den gesamten Aufbau des Werkzeugs werden diese Forderungen als erstes bei dem Systementwurf berücksichtigt. Werden Aufgaben des **dbEngineers** in Funktionen gegliedert, so ist eine Aufteilung in physische Softwaremodule entsprechend ihrer Aufgaben möglich.

---

[1] False (engl.): Falsch bzw. Nein für eine binäre Zustandsanzeige

[2] True (engl.): Wahr bzw. Ja für eine binäre Zustandsanzeige

In der Softwareentwicklung hat sich dafür das System einer flexiblen Architektur von mehrschichtigen Anwendungen (Multi-Tier[1]-Anwendung) durchgesetzt. Das Prinzip der mehrschichtigen Anwendung zeigt die nachstehende Abbildung 5-6 am Beispiel einer typischen dreischichtigen Applikation [19].

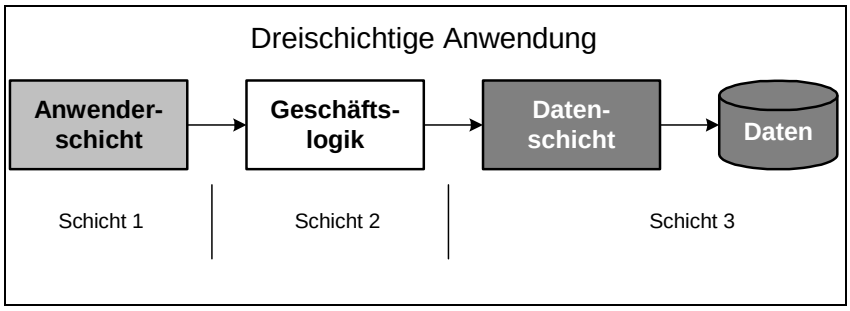

Abbildung 5-6: Aufbau einer dreischichtigen Anwendung

Bei einer mehrschichtigen Anwendung kann jede der Ebenen auf einem eigenen Rechner ausgeführt werden, wenn sichergestellt ist, dass die Ebenen sich untereinander über ein Netzwerk erreichen können (verteilte Anwendung). Welche Aufgaben die einzelnen Schichten einer mehrschichtigen Anwendung üblicherweise übernehmen, erläutert Tabelle 5-1 am Beispiel einer dreischichtigen Architektur (3-Tier).

Schicht	Aufgabe
Datenschicht	Datenspeicherung und Sicherstellung der Datenintegrität durch ein DBMS
Geschäftslogik	Implementierung der Programmlogik, des Ablaufs einzelner Arbeitsvorgänge und Prüfung auf Einhaltung aller Regeln zur Plausibilität
Anwenderschicht	Implementierung der Benutzeroberfläche

Tabelle 5-1: Aufgaben der Ebenen einer dreischichtigen Anwendung

In Multi-Tier-Anwendungen ist der Einsatz eines einzigen Objekts, das für alle Aufgaben zuständig ist, die Ausnahme. Der Vorteil der mehrschichtigen Anwendungen liegt darin, dass mehrere Objekte arbeitsteilig die Aufgaben gemeinsam erledigen. Durch das Aufteilen auf verschiedene Objekte wird jedes Objekt für sich entsprechend seiner Aufgaben optimal konfiguriert und implementiert. Werden an eine Anwendung neue Forderungen gestellt, so können diese mit neuen zusätzlichen Objekten, die bei der Anwendung angemeldet werden, erfüllt werden. Ein Beispiel für die Funkti-

[1] Multi-Tier (engl.): Mehrschichtig

onserweiterung ist das bei Microsoft Officeanwendungen bekannte Prinzip der *Add-Ins*[1], wodurch der Endanwender oder Dritte als Dienstleiter die Möglichkeit haben, ergänzende Module zu erstellen und einzubinden [21].

Der Aufbau des dbEngineers

Bei einer mehrschichtigen Anwendung hat der Anwender die Möglichkeit, die Installation und Einrichtung entsprechend seiner Leistungsanforderungen anzupassen. Es wird nicht zwangsläufig gefordert, dass alle Schichten der Anwendung auf getrennten Rechnern ablaufen müssen. Es lassen sich auch alle Schichten auf einem Einzelplatzrechner betreiben.

Die daraus entwickelte Aufteilung des **dbEngineers** in die verschiedenen logischen und physischen Software- und Datenbankmodule zeigt Abbildung 5-7. Die Anwenderschicht ist abgesetzt von den Adaptern und Modulen, die eine Zwischenschicht bilden, um auf die unterlagerte Schichten *Externe Anwendungen* und *Datenbanken* zugreifen zu können

Bezogen auf die Aufgabenstellung des **dbEngineers**, teilen sich die Funktionen in folgende Bereiche auf:

1. *Die Anwenderschicht*: Sie übernimmt im Wesentlichen die Präsentationsschicht, die Benutzerinteraktionen, die Aufrufe zur Engineeringunterstützung durch die Adapter und die Verwaltung von Modulen.
2. *Die Datenbankmodule*: Diese regeln den Zugriff auf die Datenbanken des Systems:
    a. Der projektbezogenen Datenbank
    b. Der Moduldatenbank, die projektunabhängig und unternehmensweit relevant ist.
3. *Die Adapter zu den Anwendungen*: Die Adapter ermöglichen der Anwenderschicht den Zugriff auf die externen Projektierungswerkzeuge. Sie enthalten die angesprochenen Funktionen zur Engineeringunterstützung, speziell angepasst auf das jeweilige Entwicklungswerkzeug.

[1] Add-in (engl.): Software-Erweiterungsmodul für Microsoft Officeanwendungen

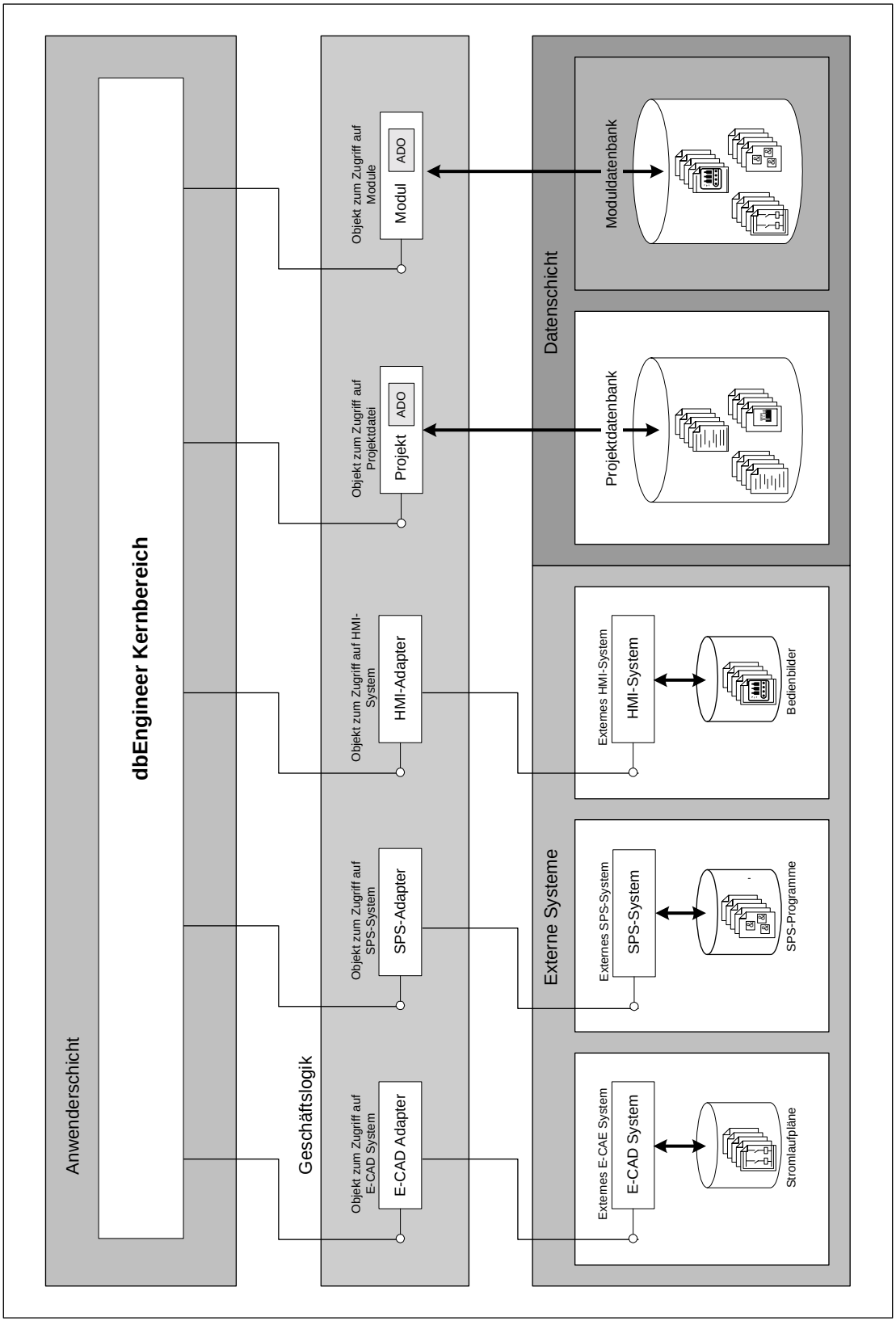

Abbildung 5-7: Architektur des **dbEngineers**

Neben den Funktionen und Zugriffsmodulen kommt der Speicherung von Daten eine besondere Bedeutung zu. Durch die Abbildung der Aspekte einer Anlage in einer Datenbankstruktur wird die Projektierungsunterstützung und automatisierte Bearbeitung erst ermöglicht. Bei dem Aufbau des **dbEngineers** werden den Forderungen nach einer funktionsbezogenen Aufteilung der Anlage in elektro-mechanische Grundsysteme mit wieder verwendbaren Projektierungsmodulen (SW-Module, Makros etc.) Rechnung getragen.

Die Daten und Informationen, die die Zerlegung der Anlage in Grundsysteme darstellen, werden in einer projektbezogenen Datenbank gespeichert. Für jedes im Unternehmen zur Bearbeitung anstehende Projekt wird eine eigene projektbezogene Datenbank auf dem DMBS angelegt.

Dagegen repräsentiert die Moduldatenbank das firmenspezifische Projektierungswissen für Grundsysteme und existiert auf dem DBMS nur einmal. Jede Anwendung **dbEngineer** muss sich daher zur Durchführung der Projektierungsunterstützung dieser Datenbank bedienen.

Der **dbEngineer** stellt dem Anwender über die Datenbankmodule den Zugriff auf die beiden Datenbanken bereit:

1. Die Projektdatenbank
2. Die Moduldatenbank.

Für den Aufbau von mehrschichtigen Anwendungen werden zunehmend Komponentenmodelle eingesetzt. Typische Vertreter von Komponentenmodellen sind derzeit CORBA[1], COM/COM+ und dessen Nachfolger .net™. Grundsätzlich eignen sich alle drei Modelle für die Aufgabe, doch auf Grund der weiten Verbreitung der COM/COM+ Schnittstelle bei den Entwicklungssystemen der Automatisierungstechnik wird COM/COM+ gewählt [33]. Die Wahl von COM/COM+ statt .net™ ist kein Nachteil, denn COM-basierende Softwarebausteine werden auch zukünftig unterstützt und sind vollständig kompatibel zu Microsoft .net™. Es wird erwartet, dass beide Technologien über einen längeren Zeitraum parallel existieren und ineinander aufgehen werden [34].

Die Wahl von COM/COM+ bietet den weiteren Vorteil, dass automatisch die Datenbankzugriffskomponenten ADO des Betriebssystems Windows genutzt werden können. Durch die

[1] CORBA Abk. für **C**ommon **O**bject **R**equest **B**roker **A**rchitecture (engl.): Plattformübergreifende Komponentenarchitektur veröffentlicht von der herstellerunabhängigen Object Management Group (OMG).

Nutzung von systemimmanenten Funktionen und Komponenten des Betriebssystems verringert sich der Installations- und Verwaltungsbedarf der Software erheblich. Gleichzeitig sinkt die Lernkurve des Anwenders in Bezug auf Administration und Verwaltung des Systems, da er auf für ihn bereits bekannte und eingeführte Strukturen, wie zum Beispiel bei der Rechtevergabe im Netz durch Windows 2000 Benutzerkonten, zurückgreifen kann.

Skalierung des dbEngineers

Durch die Wahl einer komponentenbasierten Architektur kann der **dbEngineer** flexibel für unterschiedliche Einsatzumgebungen skaliert und installiert werden. Entsprechend der Anwenderanforderungen ist die Installation der kompletten Anwendung auf einem Einzelplatzrechner ebenso möglich, wie die Einrichtung als mehrschichtige verteilte Applikation.

Eine Vielzahl der Automatisierungslieferanten sind Klein- und Mittelständische Betriebe [3], die nicht den vollen Leistungsumfang einer mehrschichtigen Architektur mit getrennten Datenbankservern benötigen. Stattdessen erwarten diese Unternehmen Einzelplatzinstallationen auf tragbaren Computern, die das Engineering auf den Anlagenbaustellen beim Investor ermöglichen.

Der **dbEngineer** lässt für sich die verschiedenen aufgeführten Szenarien konfigurieren und einsetzen. Im Folgenden sollen einige Möglichkeiten der Installation dargestellt werden.

Die verschiedenen Szenarien nach Abbildung 5-8 stellen mögliche Installationsvarianten des **dbEngineers** dar:

1. *Einzelplatz:* Der **dbEngineer** mit seinen Adaptern, das DBMS für Modulbibliothek und Projektdaten, sowie die Planungswerkzeuge befinden sich auf einem Rechner. Das ist eine typische Konstellation für ein Kleinunternehmen oder eine Inbetriebnahme.

2. *Teaminstallation:* Die Installation ist für Planungsgruppen geeignet, die Anforderungen an die Performance des Systems sind höher als im ersten Fall. Die Adapter werden deshalb zusammen mit dem DBMS auf einem eigenen Server installiert.

3. *Abteilungsinstallation:* Wegen der nochmals gesteigerten Anforderungen, laufen in diesem Fall die Adapter vom DBMS getrennt auf einem eigenen Applikationsserver.

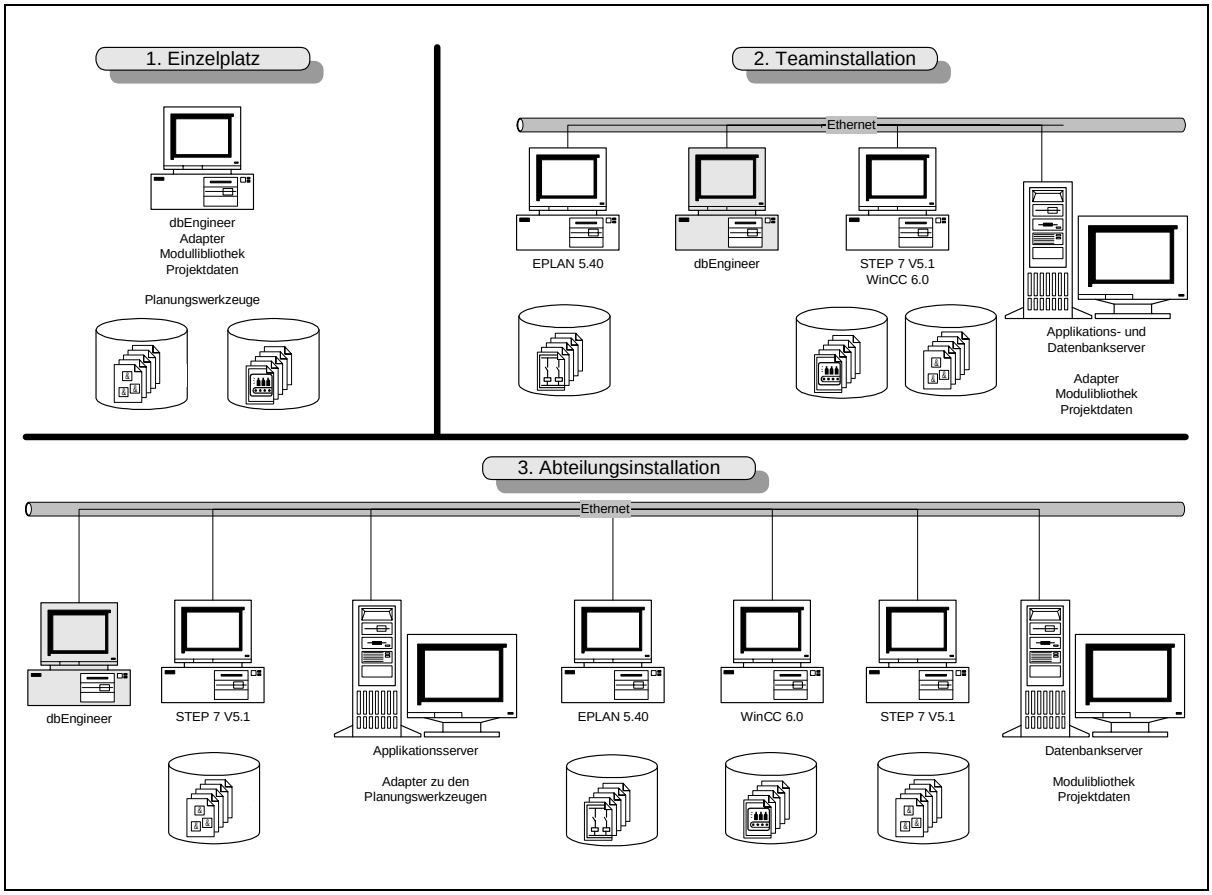

Abbildung 5-8: Szenarien zur Installation des **dbEngineers**

Die Beispiele zeigen nur einen Ausschnitt der Möglichkeiten zur Skalierung. Für Hochleistungsanwendungen ist auch die getrennte Einrichtung von Modulbibliothek und Projektdatenbank auf jeweils eigenen DBMS-Servern, sowie der Einsatz mehrerer Applikationsserver für Adapter zur Lastaufteilung realisierbar. Der **dbEngineer** kann mit der gewählten Softwarearchitektur für ein breites Anwendungsspektrum skaliert und eingesetzt werden.

Die Entwicklung der einzelnen Schichten der Multi–Tier-Anwendung **dbEngineer** aus dem Grundkonzept wird in den folgenden Kapiteln gezeigt.

5.3 Konzept der Datenbanken für den dbEngineer

Die Ablage der Informationen über die elektro-mechanischen Grundsysteme einer Anlage in einer Datenbankstruktur ist das Fundament auf dem sich das Gesamtsystem **dbEngineer** abstützt. Auf Grund seiner Bedeutung soll die Erläuterung des Aufbaus vom **dbEngineer** mit der Herleitung eines geeigneten Datenbankmodells beginnen.

Um eine Datenbank aufbauen zu können, muss zuerst ein so genanntes Datenbankmodell definiert werden, das ein möglichst exaktes Abbild der realen Welt ist. Das Datenmodell entsteht, indem durch eine Analyse der realen Welt ein zu modellierender Teilausschnitt der realen Welt definiert wird. Der modellierte Teilausschnitt heißt Diskursbereich oder Miniwelt.

Beim Betrachten der Miniwelt *Automatisierung* treten bestimmte Objekte, auch Entities[1] genannt, wie beispielsweise Aktor, Zubehör oder Design auf. Zwischen diesen Entities existieren Beziehungen, die bestimmte Abläufe oder Abhängigkeiten in der Miniwelt repräsentieren. Um zu einem Datenmodell zu gelangen, werden gleichartige Objekte und gleichartige Beziehungen jeweils zu Klassen zusammengefasst. Die Art der Darstellung von Beziehungen zwischen Objekten einer realen Welt wird Entity-Relationship-Model[2] (ER-Modell) genannt [25].

5.3.1 Datenbankentwurf für die Projektdaten

Für die Modellierung einer Anlage mit Hilfe eines Datenbankmodells kann auf die Strukturierungsvorgaben der DIN EN 61346-1 zurückgegriffen werden. Die Norm und die erläuternden Beiblätter[3] Beiblatt 1 zur DIN EN 61346-1 [35] und Beiblatt 2 zur DIN EN 61346-1 [36] beschreiben die Strukturierungskonzepte einer Anlage, deren Dekomposition in Objekte, die Aspekte eines Objektes und deren Überführung ineinander.

Wegen der Konzeption des Systems als Out-of-the-Box[4]-Anwendung soll der Anwender keinen Einfluss auf die Struktur und Gestaltung der Datenbank haben. Da die Struktur durch die Aufgabenstellung und durch die Vorgaben der DIN EN 61346 geregelt ist, bedeutet die Nutzung eines standardisierten Datenbankmodells keinen Nachteil. Der Anwender erhält die Möglichkeit über Konfigurationswerte im Rahmen der Programmbedienung auf bestimmte Einstellungen gemäß seiner Projektierungsaufgabe Einfluss auf die Datenbank zu nehmen. Die Möglichkeiten dazu werden zu einem späteren Zeitpunkt erläutert.

---

[1] Entities *pl.*, Entity (engl.): Entität

[2] Entity-Relationship-Model (engl.): Entitäten-Beziehungsmodell

[3] Die Beiblätter zur Norm sind kein Bestandteil der Norm, sondern enthalten offizielle Informationen zur zugehörigen Norm

[4] out of the box (engl.): Einsatzfertige Programme, die ohne weiteren oder mit nur sehr geringem Anpassaufwand direkt nach der Installation eingesetzt werden können

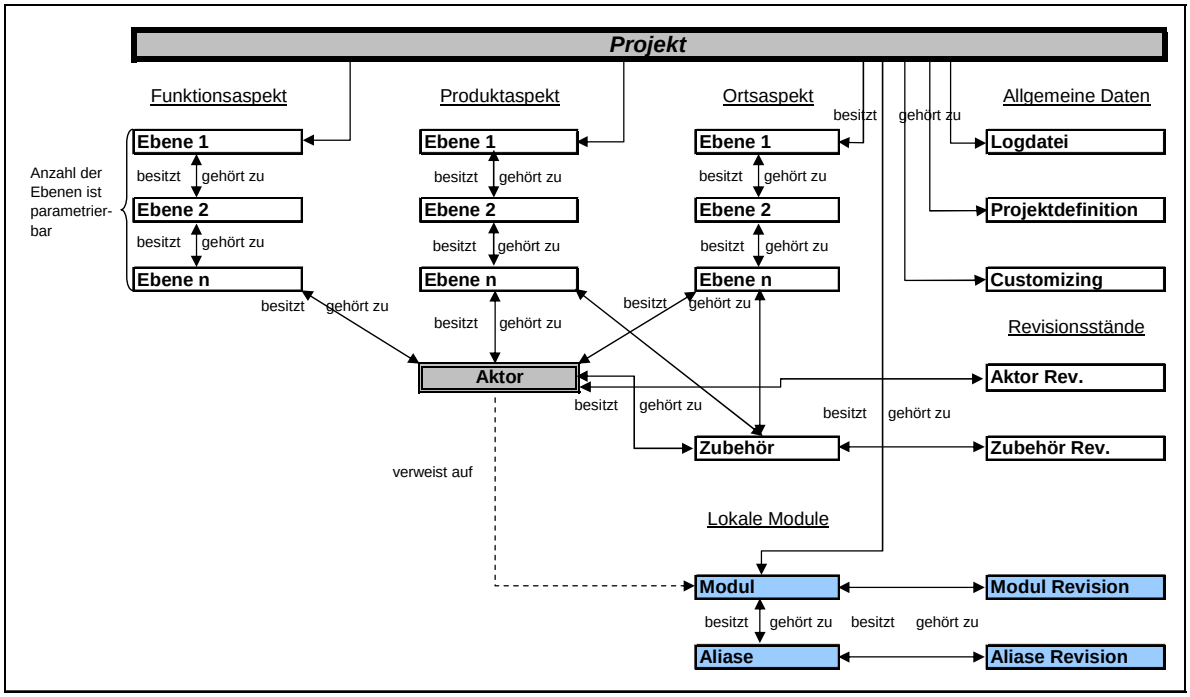

Abbildung 5-9: Entity-Relationship-Modell für die Projektdaten

Das Entity-Relationship-Modell in Abbildung 5-9 zeigt die Umsetzung der Anforderungen gemäß DIN EN 61346-1. Die zentralen Objekte *Aktor* und *Zubehör* können über alle drei Aspekte erreicht und zurückverfolgt werden. Das *Zubehör* ist beim Funktionsaspekt dem *Aktor* zugeordnet (vgl. hierzu Seite 17: Abbildung 2-11: *Anlagenstruktur als Funktionsaspekt*), während bei den anderen Aspekten *Aktor* und *Zubehör* gleichwertig an die Ebenen angebunden sind.

Die Tiefe der Strukturierung (Anzahl der Ebenen) kann für jeden der Aspekte einzeln durch den Anwender entsprechend seiner Anlagengröße bzw. Anlagenstruktur gewählt werden. Der Standardwert beträgt sechs Ebenen, die zulässige Variationsbreite ist von 4 bis 10 Ebenen einstellbar. In der Praxis hat sich eine Gliederung mit sechs Ebenen auch für komplexe Anlagen als sinnvoll herausgestellt [14]. Werden alle alphanumerischen Zeichen in den sechs Gliederungsebenen voll ausgenutzt, sind mehr als 2 Milliarden Benennungen pro Aspekt möglich, was für die Modellierung selbst großer Anlagen, wie eines Hochofens, ausreichend ist.

In der Projektdatenbank werden lokale Kopien der für das Projekt benötigten Module abgelegt. Mit den lokalen Kopien der Tabellen *Modul* und *Aliase*, wird das einzelne Projekt von den Revisionsständen der Module abgekoppelt, was für die Konsistenz der Projektierungslösungen wichtig ist. Projekte haben in der Automatisierung Durchlaufzeiten die typischerweise zwischen drei Mona-

ten und einigen Jahren liegen können. Es ist einsichtig, dass die Module der zentralen unternehmensweiten Modulbibliothek, die anschließend vorgestellt wird, in dieser Zeit Änderungen erfahren werden. Würde in einem lang laufenden Projekt ausschließlich mit der zentralen Bibliothek gearbeitet, so würde der erste Planungsabschnitt mit einer Variante A der Module generiert werden, während ein später erzeugter Planungsabschnitt mit einer revisionierten Variante B erzeugt würde. Ein solcher Sprung in der Projektierung ist weder für das Planungsteam noch für den Investor akzeptabel. Deshalb werden die benötigten Module einmalig ermittelt, aus der zentralen Bibliothek in das Projekt kopiert und die Generiervorgänge des **dbEngineers** mit der lokalen Kopie ausgeführt.

Der **dbEngineer** bietet Funktionen zum Abgleich der lokalen Ablage und der zentralen Bibliothek an, um gezielt revisionierte Module in die lokale Ablage zu kopieren und damit ältere Versionen zu ersetzen. Es soll nicht verhindert werden, dass mit den neueren Modulen gearbeitet wird, es wird aber sichergestellt, das der Projekteur die letzte Kontrolle über die verwendeten Module behält. Diese Vorgehensweise ist auch deshalb vorteilhaft, weil abgeschlossene aus dem Archiv zurückgeholte Projekte mit dem damaligen Stand der Module weitergeführt werden können, wenn zu einem späteren Zeitpunkt eine Anlagenerweiterung vorgenommen wird und keine Brüche in der Projektierung gewünscht sind.

Die im gezeigten ER-Modell dargestellten Referenzierungen werden vom **dbEngineer** beim Anlegen der Datenbank im DBMS eingetragen (SQL-2-Standard). Das DBMS entlastet den **dbEngineer** damit von Aufgaben wie zum Beispiel der Überwachung der referenziellen Integrität. Die Verlagerung bestimmter durch den SQL-2-Standard abgedeckter Funktionen in das DBMS bietet zwei Vorteile:

- *Performance*: Die datenbankinternen Abläufe sind für höchste Geschwindigkeit optimiert.
- *Entlastung*: Der **dbEngineer** wird von DB-Überwachungsaufgaben entlastet und schlanker.

Damit das Gesamtsystem **dbEngineer** datenbankunabhängig bleibt, werden weitere von den Datenbanksystemen intern bereitgestellte Funktionen, sofern sie nicht vom SQL-2-Standard abgedeckt sind, nicht eingesetzt.

5.3.2 Datenbankentwurf für die Modulbibliothek

Die bei der Projektdatenbank gemachten Aussagen über die Nutzung des DBMS gelten auch für die Modulbibliothek. Die Modulbibliothek nimmt die Musterlösungen des Unternehmens als Module auf und legt sie im DBMS gemäß folgender Struktur ab.

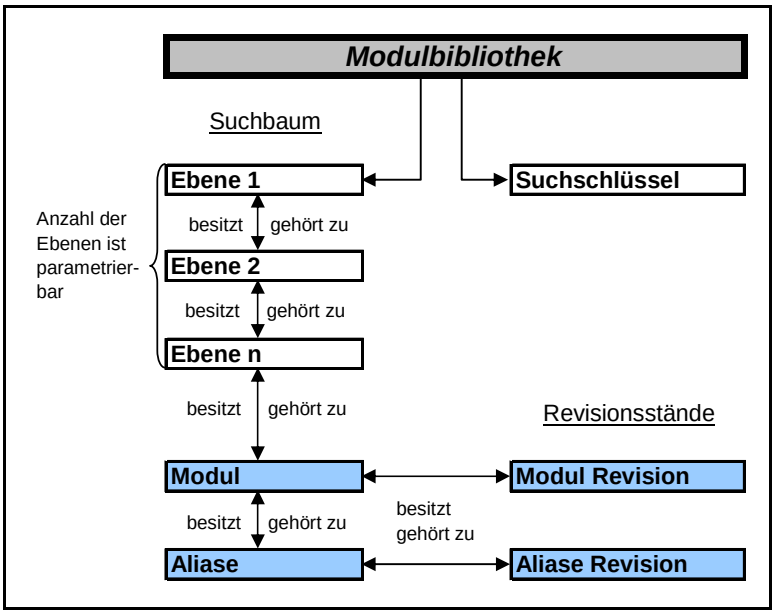

Abbildung 5-10: Entity-Relationship-Modell für die Modulbibliothek

Der Anwender nimmt auf die Struktur und den Aufbau der Modulbibliothek nur über die Gliederungstiefe des Suchbaums Einfluss, wie Abbildung 5-10 zeigt. Alle weiteren Tabellen und Beziehungen sind fest vorgegeben. Die Tabellen *Module* und *Aliase* enthalten die Daten für die, in der Projektdatenbank eingerichteten und dort diskutierten, lokalen Tabellen *Module* und *Aliase*.

Standardmäßig ist für den Suchbaum eine Tiefe von vier Ebenen eingestellt. Dabei wird ein Standard-Suchschlüssel gemäß Abbildung 5-11 für die Referenzierung von Modulen benutzt.

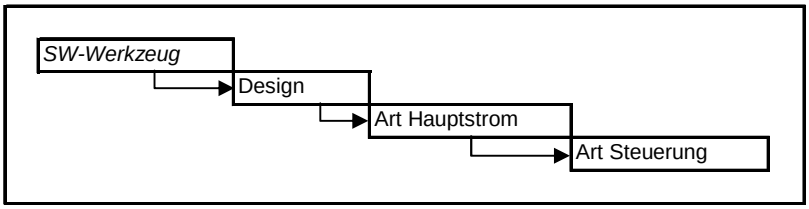

Abbildung 5-11: Bildung des Standard-Suchschlüssels für Module

Die Tiefe der Gliederung und den Aufbau des Suchschlüssels kann der Administrator festlegen. Den Aufbau des Suchschlüssels speichert der **dbEngineer** in der Tabelle *Suchschlüssel*. Felder, die zur Bildung des Suchschlüssels herangezogen werden, müssen aus den Tabellen der Projektdaten stammen. Der Suchschlüssel gilt, wie die Modulbibliothek, unternehmensweit.

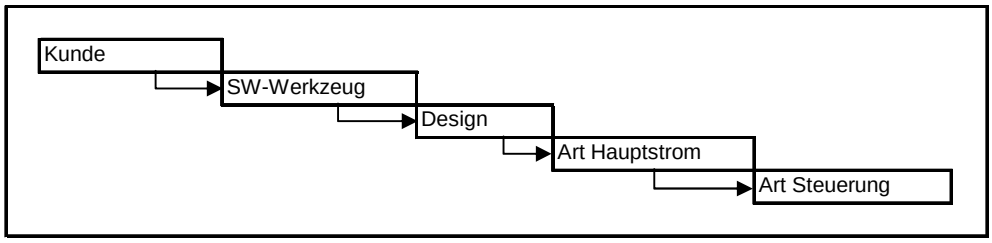

Abbildung 5-12: Individueller Suchschlüssels für Module

Die Bildung eines individuellen Suchschlüssels, wie in Abbildung 5-12 dargestellt, kann in der Praxis sinnvoll sein, wenn beispielsweise das Unternehmen Kunden verschiedener Branchen bedient, die branchentypisch angepasste Lösungen erwarten. Das Modul für eine bestimmte Klasse eines elektro-mechanischen Grundsystems unterscheidet sich je nach Kunde. Damit der **dbEngineer** die richtigen Module selektieren kann, ist es erforderlich, in den Suchschlüssel *Kunde* als Unterscheidungsmerkmal aufzunehmen.

Der Suchschlüssel wird vom DBMS auf Eindeutigkeit überwacht und in der Tabelle *Modul* im Feld *Template* gespeichert, siehe Abbildung 5-13.

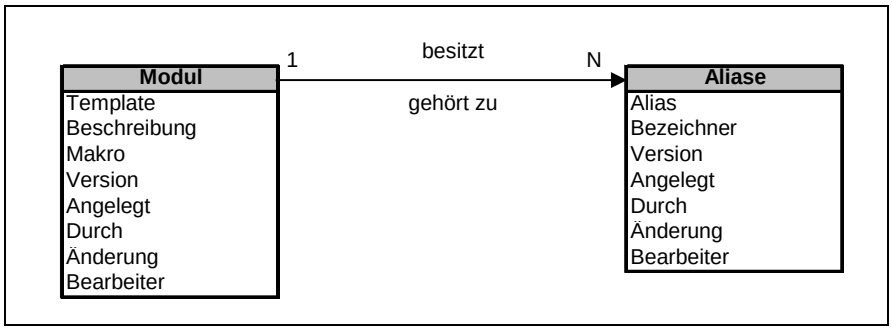

Abbildung 5-13: Detailansicht ER-Diagramm: Modul und Aliase mit Feldern

Die Module der Lösungen werden im Feld *Makro* der Tabelle *Modul* abgelegt. Das Feld Modul ist ein Feld vom Typ Binary Large Object[1] (BLOB). Datenbankfelder dieses Typs können binäre Daten von beliebiger Länge speichern. BLOB-Felder werden dazu genutzt Daten wie Bilder, Graphiken, Zeichnungen oder Videos im nativen Format abzulegen. Das DBMS stellt dabei nur den Speicherplatz für die Daten zur Verfügung, eine semantische Auswertung durch das DMBS findet nicht statt, weil es sich dabei um Daten in speziellen Formaten handeln kann. BLOB-Felder können daher auch nicht in Abfrageoperationen einbezogen werden.

Für den **dbEngineer** bietet das BLOB-Feld *Makro* die Möglichkeit die Daten eines Moduls, sei es eine Zeichnung für ein E-CAD, ein Programmteil für eine SPS oder ein Bildfenster für HMI, im nativen Format des Erzeugersystems ablegen zu können.

Beim modulbasierten Ansatz des **dbEngineers** sollen die Platzhalter der Module zur Laufzeit durch ihre absoluten Bezeichnungen ersetzt werden. Der Tabelle *Modul* ist deshalb als 1:N-Relation die Tabelle *Aliase* zugeordnet. In der Tabelle *Aliase* enthält das Attribut *Alias* den Namen des Platzhalters im Modul und das Attribut *Bezeichner* entweder eine Referenz auf ein weiteres Attribut einer Tabelle oder einen String zur Substitution des Platzhalters. *Aliase* muss über eine vollständige Liste aller im Modul benutzten Platzhalter verfügen, weil die Nachverarbeitung das Ersetzen entlang der Liste vornimmt, wie Abbildung 5-14 illustriert.

[1] **B**inary **L**arge **Ob**ject (BLOB) (engl.): Binäres großes Objekt

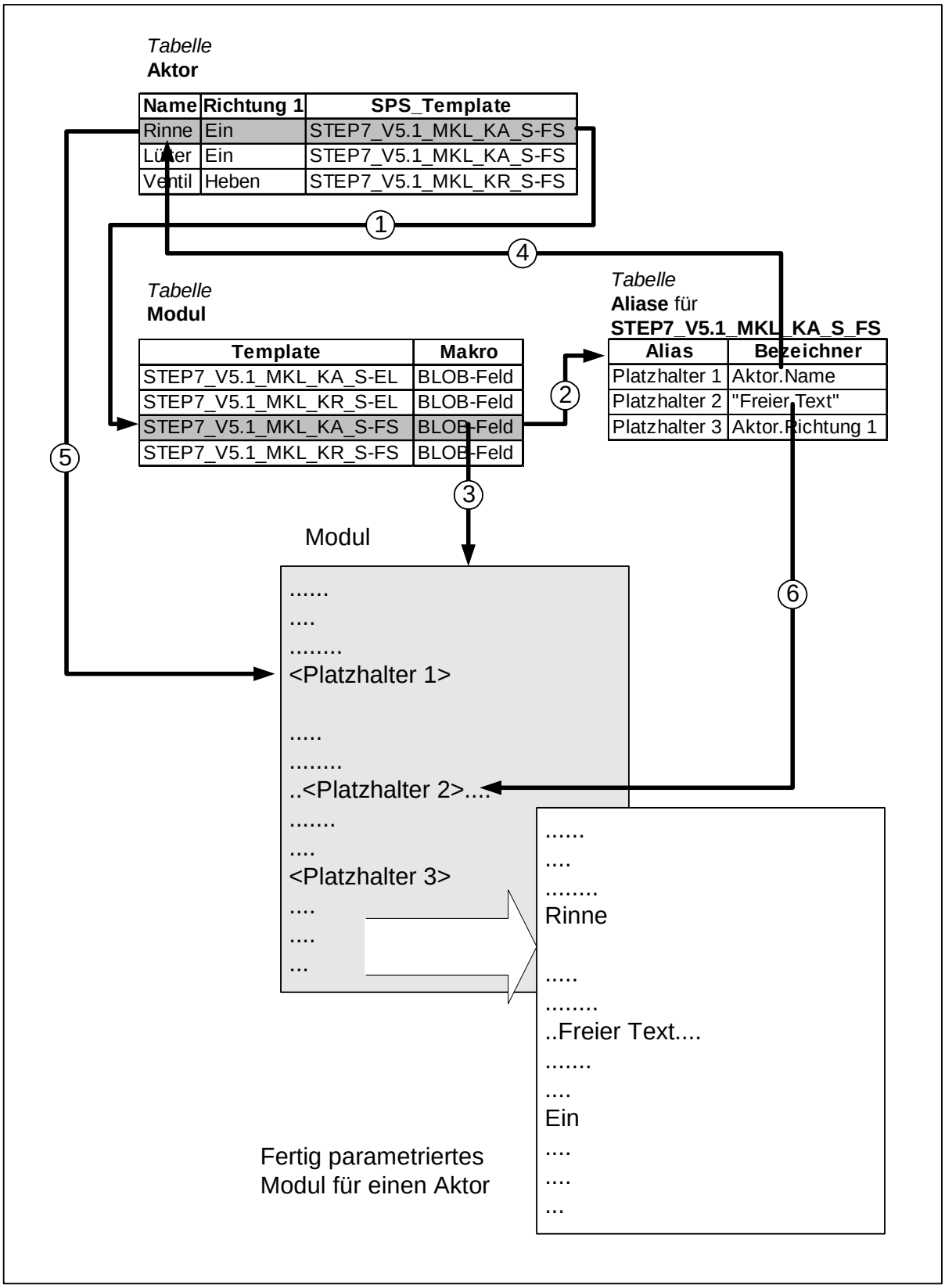

Abbildung 5-14: Prinzip der Substitution von Platzhaltern in Modulen

Der Aktor ist das zentrale Element des gesamten Vorgangs. Der dargestellte Vorgang beginnt deshalb mit dem Aktor und zeigt dann die weiteren Vorgänge am Beispiel eines SPS-Moduls. Diskutiert werden hier nur die Referenzierungen, die die Datenbankseite betreffen, wie der **dbEngineer** die Funktionen intern abwickelt, wird zu einem späteren Zeitpunkt erläutert.

1. Der Aktor, dessen Modul bearbeitet werden soll, gehört als elektro-mechanisches Grundsystem zu einer Klasse, die in *SPS_Template* spezifiziert ist. Unter der Angabe der Klasse *SPS_Template* kann in der Tabelle *Modul* über *Template* das zugehörige Modul im Feld *Makro* aufgefunden werden.
2. Auf Grund der 1:N-Relation zeigt die Detailtabelle *Aliase* die zugehörigen Platzhalter an. Im Attribut *Alias* sind die Namen der Platzhalter abgelegt, deren Originalwerte werden über *Bezeichner* angegeben. Dabei treten zwei unterschiedliche Varianten auf:

    a. *Bezeichner* enthält die Referenz auf ein weiteres Datenbankfeld,
    b. *Bezeichner* enthält einen Text, der direkt eingetragen wird.

3. Zunächst wird aus dem Feld *Makro* das Modul zur Bearbeitung geladen.
4. Das Ersetzen der Platzhalter beginnt mit der ersten Zeile der Tabelle *Aliase*. Der Inhalt von *Bezeichner* lautet *Aktor.Name*, für die Substitution heißt das, vor dem Gliederungspunkt ist der Name der Tabelle, nach dem Punkt das Attribut der Tabelle angegeben. Hier zeigt *Bezeichner* auf den Inhalt des Feldes *Name* der Tabelle *Aktor*.
5. Jedes Vorkommen von *Platzhalter 1* im Modul wird gegen *Rinne*, den Inhalt des Feldes *Name*, ausgetauscht.
6. Die nächste Zeile von *Aliase* besitzt als Feldwert *Bezeichner* einen String, gekennzeichnet durch Anführungszeichen. Jedes Vorkommen von *Platzhalter 2* kann im Modul direkt durch den Inhalt ″Freier Text″ ersetzt werden.

Die beschriebenen Vorgänge werden solange wiederholt, bis alle in *Aliase* aufgeführten *Alias* mit ihren Originalwerten ausgetauscht sind.

5.3.3 Internes XML-Format

In den vorangegangenen Kapiteln wurde der Aufbau der Datenbanken für das System erläutert. Die in den Datenbanken abgelegten Informationen müssen innerhalb des **dbEngineers** zwischen den einzelnen Objekten transportiert werden. Bei den Informationen handelt es sich um Datenmengen, deren Größe und Anzahl von Entitäten unbestimmt ist, beispielsweise ist die Antwortmenge einer SQL-Abfrage veränderlich oder die Anzahl von Zubehör eines *Aktors* variabel.

Wenn Dritte für den **dbEngineer** Module entwickeln und bereitstellen sollen, muss das Format mit dem Datenmengen übertragen werden, allgemein akzeptiert und bearbeitbar sein. Für den **dbEngineer** wird deshalb für den Austausch von Datenmengen das XML-Format gewählt, weil es einerseits flexibel ist für die Anzahl der abgelegten Datensätze und andererseits 1:N-Relationen abbilden kann. Die verschiedenen Programmiersprachen enthalten Klassenbibliotheken für den Umgang mit XML-Formaten oder können damit nachgerüstet werden. Die Objekte der Klassenbibliotheken stellen Methoden, mit denen Datenmengen aus Datenbankverbindungen in XML transformiert und rückgewandelt werden können, bereit. Die Datenzugriffkomponenten ADO bieten ebenfalls Funktionen zur XML-Wandlung an [19].

Für die Nutzung im **dbEngineer** wird folgender Aufbau für XML-Dokumente festgelegt.

```
<?xml version="1.0" standalone="yes"?>
<DATAPACKET Version="2.0">
  <METADATA>
    <FIELDS>
      <FIELD attrname="AAA" fieldtype=" "/>
      <FIELD attrname="BBB" fieldtype=" "/>
    </FIELDS>
  </METADATA>
  <ROWDATA>
    <ROW AAA=" " BBB=" "/>
  </ROWDATA>
</DATAPACKET>
```

Abbildung 5-15: Definition für alle internen XML-Dokumente

Gemäß Abbildung 5-15 enthält das Dokument zur Modellierung seiner Datenmengen innerhalb der Struktur <*DATAPACKET*> zwei Substrukturen <*METADATA*> und <*ROWDATA*>. Mit <*METADATA*> werden die benutzten Feldtypen definiert, die Domänen. Die Entitäten oder Datensätze der Datenmenge werden in <*ROWDATA*> abgelegt. Im Beispiel sind zwei Felder *AAA* und *BBB* definiert worden.

Für jedes zu definierende Feld wird eine eigene Zeile *FIELD* angelegt. Der Datentyp des Feldes wird über *fieldtype* deklariert. Die für den **dbEngineer** erforderlichen Datentypen, die über *fieldtype* deklariert werden, führt Abbildung 5-16 auf.

Datentyp	Fieldtype	Zusatzangabe	Kommentar
Datum	"date"		
Integer	"i4"		4-Byte Integer (32-Bit)
Logisch	"boolean"		Binärer Zustand: True/False
Memo	"bin.hex"	SUBTYPE="TEXT"	Textfeld mit beliebiger Länge
Real	"r8"		8-Byte Realzahl (64-Bit)
String	"string"	WIDTH =" x"	Stringlänge x: 1-255

Abbildung 5-16: Im **dbEngineer** benutzte Datentypen für XML-Dokumente

Für jeden Datensatz wird im Block <*Rowdata*> eine eigene Zeile <*Row*> erzeugt, in der die Inhalte des Datensatzes erscheinen.

Wegen der Aufteilung des **dbEngineers** in verschiedene Module gemäß dem Komponentenmodell COM/DCOM, muss für den Informationstransport zwischen den Modulen ein COM-konformer Datentyp gewählt werden. Für den Austausch der XML-Dokumente wird der Typ *Bstr* gewählt, weil er eine Stringvariable abdeckt, die keiner Längenbegrenzung unterliegt. Zudem können Datenmengen einfach in XML-Dokumente von *String*, bzw. von *String* in XML gewandelt werden, wie das nachstehende Codefragment in Abbildung 5-17 illustriert.

```
...
Var
 MyXML_1,
 MyXML_2: string;

begin
  ...
  DatasetProvider.Dataset := ADOTable;           // ADO-Tabelle mit Provider verbinden
  ClientDataSet.ProviderName := DatasetProvider; // Provider mit Datenmenge verbinden
  ...
  ...
  // Datenmenge und damit der ADO-Tabelle ein XML-Dokument zuweisen
  ClientDataSet.XMLData := MyXML_1;
  ...
  ...
  // ADO-Tabelle auslesen und in einem XML-Dokument ablegen
  MyXML_2 := ClientDataSet.XMLData;
  ...
end;
```

Abbildung 5-17: Datenmengen und XML (Borland Delphi)

Borland Delphi bietet für die Bearbeitung von Datenmengen die allgemeine Komponente *ClientDataSet*, die mit Datenbankkomponenten wie ADO zusammenarbeitet. Die Komponente *ClientData-*

Set liest und schreibt über ihre Eigenschaft *XMLData* XML-Dokumente in eine Variable vom Typ *String.* Die im Beispiel gezeigten Zugriffe über *ClientDataSet* wirken sich direkt auf die unterlagerten Datenmengen der ADO-Tabelle aus.

Mit der Kombination aus XML-Dokument und der Übergabe als Stringtyp verfügt der **dbEngineer** über ein flexibles, einfach zu handhabendes und allgemein akzeptiertes Datenformat, das auch für die Nutzung durch Dritte geeignet ist.

5.3.4 Datenbankschnittstelle

Bei einem Client/Server-Programm wird die Verbindung zur Datenbank in der Regel beim Programmstart hergestellt und die ganze Zeit offen gelassen. Der Grund dafür besteht darin, dass der Verbindungsaufbau ein relativ zeitaufwendiger und ressourcenintensiver Vorgang ist. Beim Verbindungsaufbau sind in der Regel zwei getrennte Rechner beteiligt, sodass die Antwortzeit des Netzwerks eine Rolle spielt.

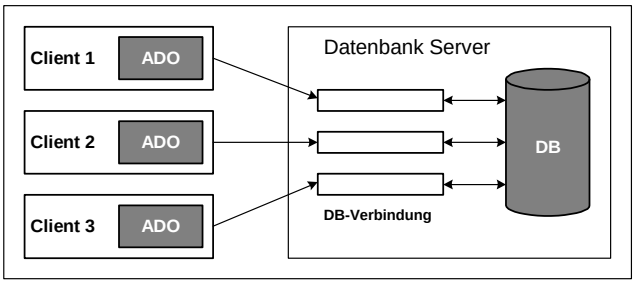

Abbildung 5-18: Aufbau einer Client/Server-Verbindung [21]

Außerdem fordert der Client wie in Abbildung 5-18 ersichtlich, direkt beim DBMS eine Verbindung an, sodass der SQL-Server die Zugriffsberechtigungen des Clients nachprüfen kann. Jeder Client belegt damit eine Verbindungsressource, was einerseits bei einer Vielzahl von Anwendern Perfomanceverluste des Servers verursacht und andererseits den Zugang beschränkt, wenn das DBMS seine Lizenzen pro Verbindung berechnet und nur eine begrenzte Anzahl von gleichzeitigen Nutzern zugelassen ist. Nicht genutzte Datenbankverbindungen werden deshalb getrennt, womit der Client eine Ressource frei gibt, die er nicht nutzt. Der Nachteil ist der Zeitverlust, der jedes Mal beim Verbindungsaufbau verursacht wird, wenn der Client die Verbindung wieder benötigt.

Bei einer verteilten Anwendung, bei der die Anzahl der gleichzeitig aktiven Benutzer unbestimmt ist, wird dies zum Problem. Zum einen begrenzen Server die maximale Anzahl der gleichzeitig aktiven Verbindungen und zum anderen kann auch der Applikationsserver nur eine begrenzte Anzahl von Objekten gleichzeitig ausführen. COM+ löst dieses Problem, indem die Objekte über Just-in-Time Activation[1] (JITA) nur für den Zeitraum des Client-Zugriffs leben und die Datenbankverbindungen über einen Pool gemeinsam genutzt werden. Wenn der Client eine Interface-Methode des auf dem Applikationsserver ausgeführten Objekts aufruft, greift das Objekt nur für den Zeitraum dieses Methodenaufrufs auf eine freie Datenbankverbindung aus dem Pool zu. Wenn das aufgerufene Objekt die ADO-Verbindung wieder trennt, bleibt die Datenbankverbindung zum Server aktiv, da die nicht mehr benötigte aktive Verbindung zurück in den Pool gelegt wird. Eine Datenbankverbindung wird somit im Zeitmultiplex-Verfahren für COM+ für mehrere Clients gemeinsam nutzbar. Daraus ergibt sich, dass eine deutlich höhere Anzahl von Clients die beschränkt zur Verfügung stehende Ressource Datenbankverbindung nutzen kann. Die Arbeitsweise des Datenbankverbindungspools zeigt Abbildung 5-19.

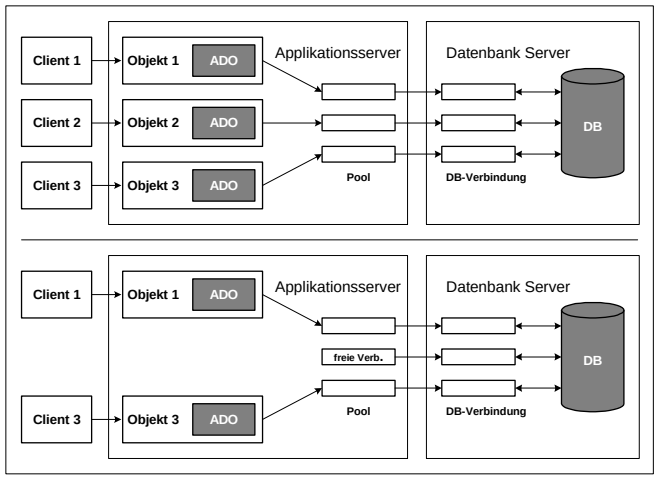

Abbildung 5-19: Datenbankverbindungspool: Oben mit drei Clients, unten nach Abmelden von Client 2 und Freigabe des Objektes [21]

Der Pool kann jedoch nur dann effektiv arbeiten, wenn alle Datenbankverbindungen mit gleichen Benutzerrechten auf den Server zugreifen, wegen des Multiplexverfahrens ist eine eigene Rechtevergabe für jeden Client nicht mehr möglich. Alle Objekte greifen mit den gleichen Rechten auf das

[1] Just-in-Time Activation (engl.): zum benötigten Zeitpunkt aktivieren

DBMS zu. Die Rechtevergabe wird deshalb bei COM+ mit Hilfe der Komponentendienste über die Objekte verwaltet. COM+ ermöglicht es für jedes Interface und jede Methode eines Objektes eigene Rechte zur Ausführung zu vergeben. Von dieser Option wird bei den Modulen für den Datenbankzugriff Gebrauch gemacht, wie im nächsten Kapitel gezeigt wird [21].

5.3.5 COM Schnittstellen des Datenbankkerns

Gemäß der vorstehenden Konventionen, dass COM-Objekte spezialisierte Aufgaben übernehmen sollen, besteht der Datenbankkern des **dbEngineer** aus den zwei eigenständigen COM-Servern *Projekt* und *Modul.* Jeder der Server ist ein für seinen Aufgabenbereich optimierter Teil der Geschäftslogik.

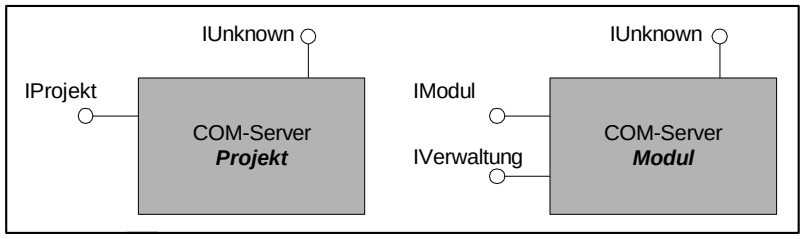

Abbildung 5-20: Darstellung der COM-Server für den Datenbankkern

Wie Abbildung 5-20 zeigt, sind die beiden Server auch bezüglich ihrer Schnittstellen unterschiedlich aufgebaut. Der COM-Server *Projekt* besitzt ein Interface:

- *IProjekt:* Das Interface bietet dem **dbEngineer** den Zugriff auf die projektabhängigen Daten.

Der COM-Server *Modul* besitzt zwei unabhängige Interfaces:

- *IModul:* Über das Interface *IModul* wird dem **dbEngineer** der Zugriff auf die unternehmensspezifischen Module ermöglicht.
- *IVerwaltung:* Das Interface dient der Verwaltung der Module in der Datenbank *Module.*

Durch die Aufteilung der Funktionen auf zwei Interfaces bei dem Server *Modul* hat der Anwender die Möglichkeit, die Zugriffsrechte auf die implementierten Methoden der Interfaces getrennt zu vergeben. Um die Rechtevergabe zu vereinfachen, sind alle Methoden für die Administration in einem eigenem Interface *IVerwaltung* zusammengefasst worden. Gemäß COM+-Konventionen, wäre die Trennung nicht erforderlich gewesen, weil für jede Methode einzeln Rechte vergeben

werden können. Es wurde aber der Aufteilung auf zwei Interfaces der Vorzug gegeben, weil damit eine zentrale Stelle zur Rechtevergabe des Interfaces *IVerwaltung* existiert.

Bei der Installation in einer Mehrbenutzerumgebung ist es von größter Bedeutung, dass die in der Datenbank hinterlegten Module zentral und für alle Nutzer gleichzeitig modifiziert werden, um Inkonsistenzen in den einzelnen Projektierungen zu vermeiden. Üblicherweise schreiben Unternehmen in ihren Qualitätsmanagementsystemen nach DIN EN 9001:2000 [37] für neue oder geänderte Module den Durchlauf von Prüf- und Testzyklen vor. Erst nach erfolgreichem Durchlauf der Tests kann für ein Modul eine Freigabe erfolgen und das Modul kann in der Datenbank zur allgemeinen Projektierung freigegeben werden. Durch eine eigenständige Rechtevergabe für das Interface *IVerwaltung* können diese Aufgaben einem Administrator übertragen werden, dem die Koordination der Modulverwaltung obliegt.

Die Rechte für die projektierungsspezifischen Methoden, die in den beiden Schnittstellen *IModul* und *IProjekt* zusammengefasst sind, müssen durch den Administrator so vergeben werden, dass der Zugriff während des allgemeinen Projektierungsablaufs gewährleistet ist.

Der Server PROJEKT

Zum besseren Verständnis der Funktionsweise des COM-Objekts *Projekt* werden einige, der für die Schnittstelle entwickelten, Methoden vorgestellt. Einige der Methoden modifizieren vorhandene Datensätze im Projekt, für sie gelten besondere Vorschriften. Alle Methoden, die vorhandene Daten verändern, arbeiten derart, dass die existierenden Daten in die zugehörigen Tabellen der Revisionshistorie verschoben (siehe *Abbildung 5-9: Entity-Relationship-Modell für die Projekt*), und erst danach Daten geändert werden. Diese Archivierungsfunktion für bestehende Daten wird bei den nachstehenden Methoden nicht mehr im Einzelnen aufgeführt.

- *CreateProjekt*. Die Methode dient zum Anlegen einer neuen projektbezogenen Datenbank auf dem Datenbankserver. Aufrufparameter sind unter anderem der Projektname als Datenbankname und die Anzahl der Gliederungsstufen für die einzelnen Aspekte. Mit dem Aufruf der Methode werden die erforderlichen leeren Tabellen auf dem DBMS angelegt, und deren referenzielle Beziehungen untereinander eingerichtet.

- *ImportMuK.* Mit dieser Methode wird der Inhalt einer MuK-Liste in die Datenbank eingetragen. Die MuK-Liste wird im definierten internen XML-Format als Variable von Typ *BStr* in der Methode übergeben. Der Import erfolgt unter Berücksichtigung und Prüfung des GUIDs. Für jede zu importierende Entität wird sichergestellt, dass ein GUID vorhanden ist. Elemente mit fehlenden GUIDs werden nicht importiert, sondern in einer Fehlerdatei (Datenfile auf einem physischen Datenträger) mit einer XML-Struktur, wie die MuK-Liste, abgelegt. Diese Daten können korrigiert und wie beschrieben importiert werden. Um den Umgang mit den Daten zu steuern, veröffentlicht das Objekt mehrere Konstante, die beim Aufruf dieser Methode *ImportMuK* benutzt werden:
  - *coMuK_Append*: Fügt die Datensätze der importierten MuK-Liste an die Zieltabelle im Projekt an.
  - *coMuK_Update*: Aktualisiert Datensätze in der Projektdatenbank mit entsprechenden Datensätzen der MuK-Liste. Die Aktualisierung erfolgt auf der Basis des GUIDs.
  - *coMuK_AppendUpdate*: Falls ein entsprechender Datensatz in der Zieltabelle existiert, wird er aktualisiert. Andernfalls werden die Datensätze an die Zieltabelle angehängt.
  - *coMuK_Overwrite*: Erzeugt die Zieltabelle auf Grundlage der Struktur der Quelltabelle. Wenn die Daten in der Zieltabelle bereits existieren, werden die Daten der MuK-Liste eingetragen.
- *ExtractMuK:* Liefert die aktuell in der Projektdatenbank eingetragenen Komponenten als MuK-Liste gemäß Definition als XML-Dokument in einer Variablen von Typ *BStr* zurück. Die als MuK-Liste zurückgegebene Datenmenge ist eine Untermenge der in der Datenbank hinterlegten Informationen über die Klassen *Aktor* und *Zubehör*.
- *SelectItem:* Mit der Methode wird die Datenmenge eines Elements der Klassen *Aktor* oder *Zubehör* zur Bearbeitung zurückgeliefert.
- *NewItem:* Diese Methode fügt ein neues Element in die Klassen *Aktor* oder *Zubehör* ein.
- *UpdateItem:* Überschreibt eine vorhandene Entität mit neuen Werten, vereinbarungsgemäß darf dabei der GUID nicht verändert werden.

- *ViewAspekte:* Zeigt die Sicht auf die Anlage, wahlweise in der Strukturierung: Funktionsaspekt, Produktaspekt oder Ortsaspekt.

Wie bereits beim Datenbankentwurf erläutert, verwaltet jedes Projekt eine lokale Kopie von benötigten Modulen. Für die Bearbeitung von Modulen sollen einige der spezialisierten Methoden vorgestellt werden:

- *CopyModule:* Die Methode dient dazu, Module aus der Modulbibliothek in der Projektdatenbank lokal abzulegen, wenn noch kein Modul gespeichert wurde. Bestehende Module werden nicht verändert.

- *UpdateModule*: Die Methode vergleicht die lokal abgelegten Module mit den Modulen der Modulbibliothek und legt neuere Versionen aus der Bibliothek lokal ab. Dabei werden vorhandene Module in die Tabellen der Revisionshistorie verschoben und dann die neuen Module gespeichert.

Die weiteren Methoden zur Modulbehandlung entsprechen den nachstehend erläuterten Methoden der Schnittstelle *IModul* des Servers *Modul*, sie arbeiten allerdings nur mit den lokalen Kopien von Modulen in der Projektdatenbank. Für Zugriffe auf die Modulbibliothek wird immer der Server *Modul* benötigt.

Die Schnittstelle IModul des Servers MODUL

Der Server *Modul* dient dem Zugriff auf die Module, die das spezifische Unternehmenswissen repräsentieren. Die Rechtevergabe für die Schnittstelle *IModul* ist so zu organisieren, dass sie für den Projekteur freigegeben ist. Zum besseren Verständnis der Funktionsweise werden einige Methoden der Schnittstelle vorgestellt:

- *FindModul:* Die Methode dient dem Auffinden eines Moduls gemäß dem Suchschlüssel. Ist der Rückgabewert *leer*, so existiert für diese Klasse noch kein Modul. Der **dbEngineer** informiert den Anwender über fehlende Module mit Hilfe der Logdatei in der Projektdatenbank. Das Modul muss noch erstellt werden. Existiert ein Modul wird es als Datenmenge in einem XML-Dokument zurückgeliefert.
- *ViewTree:* Die Methode wird zum Sichten der vorhandenen Modulbibliothek durch den Anwender genutzt.

Die Schnittstelle IVerwaltung des Servers MODUL

Für die Verwaltung von Modulen wurde ein eigenes Interface deklariert, damit ein geregelter Zugriff auf Daten der Moduldatenbank organisiert werden kann. Für die Engineeringunterstützung des **dbEngineers** wird das Interface nicht benötigt. Die wichtigsten Methoden der Schnittstelle sind:

- *NewModul:* Die Methode fügt der Moduldatenbank ein neues Modul zu.
- *UpdateModul:* Mit der Methode wird ein neuer Revisionsstand eines Moduls in die Datenbank eingetragen. Dabei wird der Vorgängerstand automatisch in die Tabelle Revision verschoben, so dass er für eine Revisionshistorie zu Verfügung steht.

5.4 Architektur des dbEngineers

Die Präsentationsschicht des **dbEngineers** wird durch eine WIN32-Anwendung repräsentiert. Die Präsentationsschicht bedient sich der Datenzugriffe auf die Datenbanken des Projekts und der Modulbibliothek externer COM+ Komponenten. Das Zusammenspiel zwischen der WIN32-Anwendung und den DB-Zugriffsobjekten ist derart eng, dass zwar im Rahmen dieser Arbeit die Schnittstellen diskutiert werden, aber sie sollen nicht von Dritten aufgerufen und für ihre Zwecke genutzt werden.

Demgegenüber sollen die als COM-Komponenten ausgeführten Adapter für die Zugriffe auf externe Anwendungen (Planungswerkzeuge) und die Import/Export-Schnittstelle auch von Dritten zugeliefert werden können. Für diese Komponenten wird eine Interfacestruktur definiert, die die Komponente einhalten muss. Aus Sicht der Präsentationsschicht handelt es sich bei den Komponenten um Add-Ins, auf die die Präsentationsschicht über die vertraglich festgelegte Schnittstelle zugreifen soll. Damit eine Komponente zur Verwendung an der Präsentationsschicht angemeldet werden kann, muss die Komponente die Spezifikationen strikt einhalten.

Der Präsentationsschicht bleibt verborgen, wie die Komponente die verlangte Implementierung durchführt, sie darf darüber auch keine Annahmen treffen. Dies gilt insbesondere für die Art und Weise, wie die Komponenten mit den externen Anwendungen zusammenarbeiten.

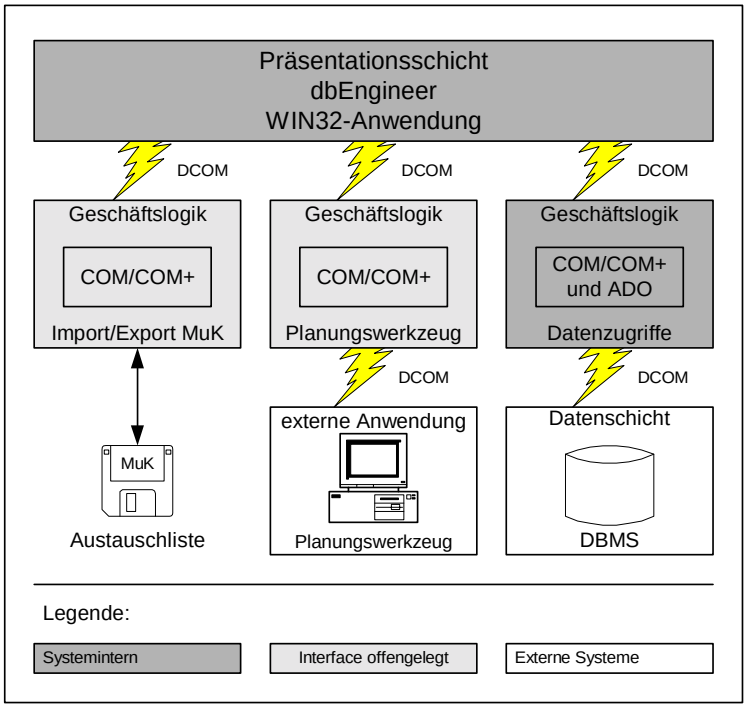

Abbildung 5-21: **dbEngineer** als mehrschichtige Anwendung

Die Mittelschicht des **dbEngineers** zeigt Abbildung 5-21. Eine Sonderrolle kommt dabei den Komponenten der Datenzugriffe zu, sie sind zwar als Komponenten ausgeführt, um die Skalierbarkeit und Modularität des Aufbaus sicherzustellen, aber sie sollen nicht durch Dritte erstellt oder aufgerufen werden. Die Geschäftslogik lässt sich demnach in drei Bereiche aufteilen, die von einander unabhängige Aufgaben wahrnehmen.

Die Aufgabenverteilung zwischen den Bereichen ist dabei wie folgt geregelt:

- *Import/Export MuK-Liste*: **dbEngineer** spezifiziert ein Interface für COM+ Anwendungen, mit deren Hilfe Dritte Komponenten zum Austausch von MuK-Listen erstellen können. Die Präsentationsschicht bietet einen Mechanismus zum Anmelden und Verwalten von Komponenten zur Laufzeit.
- *Planungswerkzeug*: **dbEngineer** spezifiziert ein Interface für COM+ Anwendungen, mit deren Hilfe Dritte Komponenten (Adapter) zum Zugriff auf Planungswerkzeuge erstellen können. Die Präsentationsschicht bietet einen Mechanismus zum Anmelden und Verwalten einer beliebigen Anzahl von Komponenten zur Laufzeit.
- *Datenzugriffe*: Enthält die COM+ Objekte zum Datenbankzugriff, wie in Kapitel 5.3.5 *COM Schnittstellen des Datenbankkerns* beschrieben.

*5.4.1 Zusammenspiel: **dbEngineer** und Adapter*

Im Zusammenspiel zwischen **dbEngineer** und den Adaptern zu den Anwendungen müssen die Probleme:

- Funktionen der Planungswerkzeuge und
- Auffinden von Adaptern und Planungswerkzeugen

gelöst werden. Der **dbEngineer** kennt zum Zeitpunkt seiner Erstellung die Planungswerkzeuge und deren Funktionen, die aufzurufen sind, nicht. Gleichzeitig kann der Anwender, durch die Möglichkeit das System zu skalieren, die Adapter und Planungswerkzeuge auf unterschiedlichen Rechnern installieren. Diese Installation darf sich auch von Projekt zu Projekt unterscheiden. Projektabhängig können sich sowohl das Werkzeug als auch dessen Standort ändern.

Wie der **dbEngineer** mit den Adaptern zu den Planungswerkzeugen zusammenarbeitet und die auf unterschiedliche Standorte verteilten Softwaresysteme verwaltet, wird in den beiden folgenden Kapiteln erläutert. Zunächst wird das Zusammenwirken mit den Adaptern dargestellt, dem schließt sich die Verwaltung von Softwaremodulen und Planungswerkzeugen durch den **dbEngineer** an.

*Funktionsaufrufe durch den **dbEngineer***

Der **dbEngineer**, der als Binärcode ausgeliefert wird, kann nicht vorab in der Softwareentwicklung für alle möglichen Planungssysteme und deren Funktionen programmiert werden. Einerseits würde die Berücksichtigung einer Vielzahl von Systemen die Entwicklung des **dbEngineer** verzögern, andererseits unterliegen die Systeme einem ständigen Wandel, so dass das System schnell veralten würde. Der Versuch, die Planungssysteme direkt im **dbEngineer** zu integrieren, wäre vergleichbar mit der Situation in der Informationsverarbeitung vor einigen Jahren, als MS-DOS-basierte Programme Druckertreiber für eine Vielzahl von Geräten in der Applikation bereitstellten. Modernere Betriebssysteme wie Windows stellen Programme nur eine durch das Betriebssystem definierte Schnittstelle zu einem Druckertreiber, den der Hersteller des Geräts liefert, bereit, wodurch das Gesamtsystem flexibler wird, weil es ohne Programmänderungen neue Geräte oder aktualisierte Treiber verwalten kann.

Mit der Entwicklung des **dbEngineers** wird ein ähnlicher Weg beschritten, der **dbEngineer** wird nicht direkt an die Planungssysteme angebunden. Stattdessen greift der **dbEngineer** über ein

Schnittstelleninterface *IAdapter* auf Adapter, die zwischen **dbEngineer** und dem Planungswerkzeug geschaltet sind, zu. Für alle Arten und Typen von Planungssystemen wird ein einziges universelles Interface *IAdapter* definiert, dessen Spezifikation von den Adaptern einzuhalten ist.

Die Adapter stellen die Verbindung zwischen der Anwenderschicht des **dbEngineers** und dem Planungswerkzeug her. Das Interface *IAdapter* definiert und spezifiziert, wie die Anwendung **dbEngineer** mit den Adaptern zu den Planungswerkzeugen kommuniziert. Die Art und Weise wie der Adapter mit seinem Entwicklungswerkzeug zusammenarbeitet ist Sache der Implementierung des jeweiligen Adapters als COM-Server. Die Implementierung wird hier nicht betrachtet, da für die Anwendung **dbEngineer** die Arbeitsweise des Adapters und dessen Zusammenarbeit mit dem unterlagerten Planungswerkzeug transparent ist. Im Kapitel 6 wird exemplarisch die Arbeitsweise eines ausgeführten Adapters erläutert. Für die Entwicklung des **dbEngineers** ist es wichtig zu betrachten, wie mit einer einzigen Schnittstelle unterschiedliche Systeme integriert werden können, ohne dass es zu Funktionseinschränkungen kommt.

Damit ein universelles Interface definiert werden kann, ist zunächst die Arbeitsweise des **dbEngineers** im Zusammenspiel mit den Adaptern festzulegen. Der Generierlauf für das Engineering wird vom **dbEngineer** für jeden Aktor einzeln angestoßen, dabei werden dem Adapter die benötigten Informationen als Parameter übergeben. Der **dbEngineer** wickelt über drei Methodenaufrufe die Generierungsunterstützung ab:

- *StartGen*: Informiert den Adapter über den Beginn eines Generierlaufs
- *Generate*: Start der Generierung für einen Aktor
- *EndGen*: Informiert den Adapter über das Ende eines Generierlaufs

In einem Flussdiagramm dargestellt, sieht der Ablauf des Generieren wie in Abbildung 5-22 gezeigt aus. Der **dbEngineer** initialisiert den Adapter durch den Aufruf von *StartGen.* Anschließend erfolgt die Generierung durch den Aufruf der Methode *Generate* für einen einzelnen Aktor. Diese Methode wird in einer Rekursion solange von **dbEngineer** aufgerufen, bis die Daten aller Aktoren bearbeitet wurden. Beim Aufruf des Generiervorgangs mit *Generate* werden vom Adapter unter anderem die Vorgänge *Finden von Modulen* und *Substituieren von Platzhalten* durchgeführt, wie mit Abbildung 5-14: *Prinzip der Substitution von Platzhaltern in Modulen* gezeigt und erläutert (Kapitel 5.3.2., Seite 84ff). Abschließend wird das erzeugte Modul durch den Adapter in das Planungssystem eingetragen.

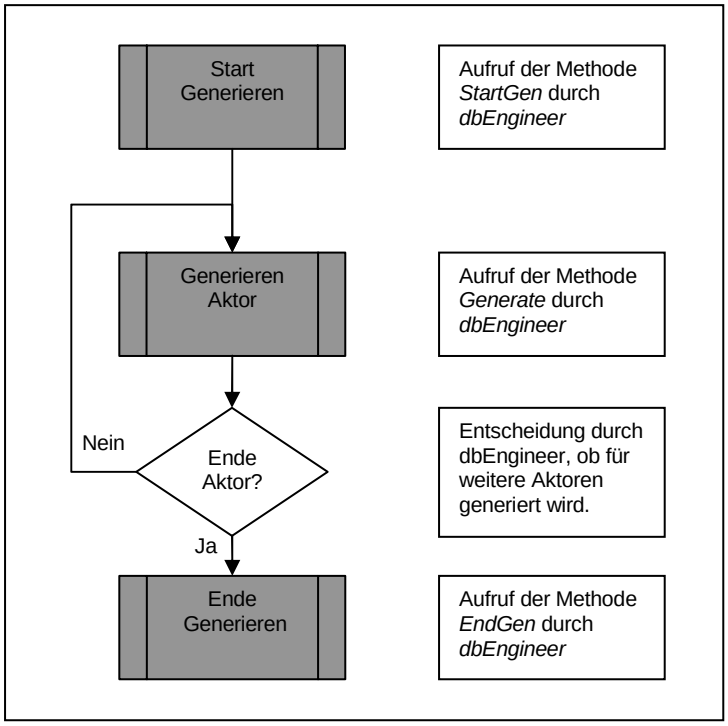

Abbildung 5-22: Flussdiagramm zum schematischen Ablauf der Funktionsaufrufe beim Generieren

Damit unterschiedliche Planungssysteme und deren verschiedene Funktionen mit nur drei Methoden angesprochen werden können, wird bei jedem der drei Methodenaufrufe ein Befehlsparameter *ID* übergeben. Der Befehlsparameter ist eine Konstante vom Typ *long*[1], die durch den jeweiligen Adapter definiert ist und dem Adapter die Auswahl der gewünschten Funktion ermöglicht.

```
procedure TAdapter.Generate(ID: Integer; const Aktor, Modul: WideString);
begin
  // Befehlsparameter dekodieren
  case ID of
    // Funktion für Befehlsparameter "1000" ausführen
    1000: DoSomething_1 (Aktor, Modul);
    // Funktion für Befehlsparameter "1001" ausführen
    1001: DoSomething_2 (Aktor, Modul);
  else ;
    // Unbekannter Befehlsparameter
    ErrorKommando;
  end;
end;
```

Abbildung 5-23: Dekodieren des Befehlsparameters ID durch eine case-Anweisung im Adapter (Borland Delphi)

[1] long (engl.): *hier:* vorzeichenbehaftete 32-Bit Ganzzahl.

Das Codefragment in Abbildung 5-23 zeigt, wie ein Adapter durch Dekodieren des Befehlsparameters *ID* unterschiedliche interne Funktionen erreichen kann. Durch Verwendung eines Befehlparameters *ID* kann der **dbEngineer** mit einer einzigen Methode eine Vielzahl von Funktionen im Adapter auslösen. Es ist Sache des Adapters den Befehlsparameter zu dekodieren und die gewünschte Funktion auszuführen.

Weil der **dbEngineer** definitionsgemäß keine Funktionen des Werkzeugs kennt oder Annahmen über den Funktionsumfang machen darf, müssen die unterstützten Funktionen des Adapters und deren Befehlsparameter beim Adapter abgefragt werden. Jeder Adapter muss eine Methode zur Abfrage des implementierten Funktionsvorrats bereitstellen. Das Interface *IAdapter* deklariert dazu die Methode *Query_ID*. Mit Aufruf der Methode *Query_ID* liefert der Adapter eine Auflistung aller internen Funktionen und deren Befehlsparameter zurück.

Bei der Liste handelt sich um eine Aufzählung mit variabler Länge, deshalb wird ein XML-Dokument *Liste* als Rückgabevariable der Methode *Query_ID* vom Typ *BStr* definiert. Für jede Funktion, die der Adapter bereitstellt, wird eine Entität mit ihren Attributen und deren Domänen[1] gemäß Tabelle 5-2 festgelegt.

Name	Typ	Länge
ID	long	
Funktion	String	30
Kommentar	string	50

Tabelle 5-2: Attribute und deren Domänen für das XML-Dokument Liste

Das daraus entwickelte XML-Dokument *Liste* besitzt definitionsgemäß einen Aufbau nach Abbildung 5-24. Die Daten der einzelnen Funktionen werden durch den Adapter in den Bereich <*RowData*> eingetragen und bilden hier eine verlängerbare Liste. Der Adapter fügt hier so viele Zeilen <*Row*> ein, wie er Funktionen bereitstellt. Struktur und Aufbau des XML-Dokumentes dürfen bei dem Einfügevorgang nicht verändert werden.

[1] Domäne: Begriff aus der Bereich der relationalen Datenbanken: Datentyp

```
<?xml version="1.0" standalone="yes"?>
<DATAPACKET Version="2.0">
  <METADATA>
    <FIELDS>
      <FIELD attrname="ID" fieldtype="i4">
        <PARAM Name="PROVFLAGS" Value="7" Type="i4" Roundtrip="True"/>
      </FIELD>
      <FIELD attrname="Funktion" fieldtype="string" WIDTH="30"/>
      <FIELD attrname="Kommentar" fieldtype="string" WIDTH="50"/>
    </FIELDS>
    <PARAMS DEFAULT_ORDER="1" PRIMARY_KEY="1" LCID="1033"/>
  </METADATA>
  <ROWDATA>
    <ROW ID=" " Funktion=" " Kommentar=" "/>
  </ROWDATA>
</DATAPACKET>
```

Abbildung 5-24: Definition für das XML-Dokument *Liste*

Für die Arbeit des **dbEngineer** wird die Auflistung der Befehlsparameter *ID* benötigt. Die Auflistung und Darstellung von implementierten Funktionen eines Adapters, die über *Query_ID* abgefragt wurden, illustriert Abbildung 5-25. Die beiden Parameter *Funktion* und *Kommentar* dienen zur Interaktion mit dem Benutzer und zum *Customizing*[1] des **dbEngineers**, wie später gezeigt wird.

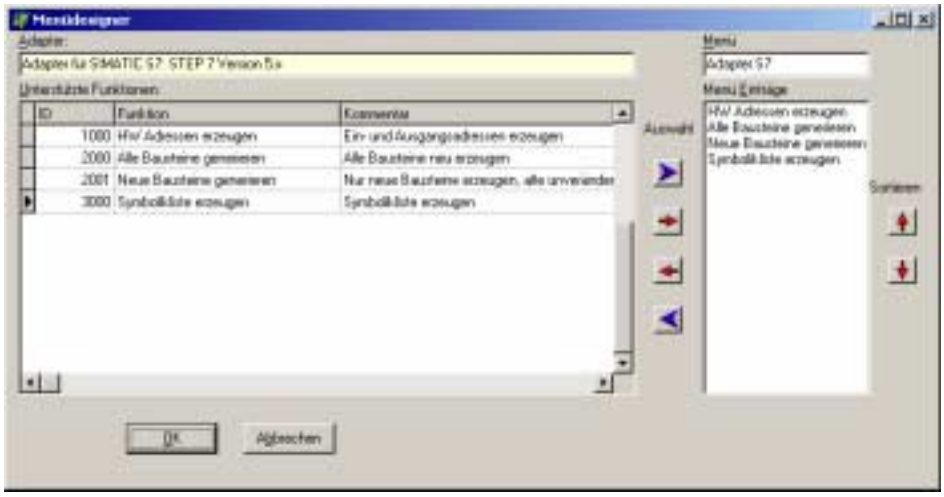

Abbildung 5-25: Mit *Query_ID* ermittelte Funktionen eines Adapters und ihre Darstellung im **dbEngineer**

Mit den Festlegungen der Methodenaufrufe und Arbeitsweise des **dbEngineers** kann jetzt die Definition des Interfaces *IAdapter* vorgenommen werden.

[1] customizing (engl.): Anpassung; hier: die Anpassung eines Programms an die Erfordernisse des Anwenders

Das Interface IADAPTER

Das für alle Adapter verbindlich geltende Interface *IAdapter* wird in der Microsoft-IDL[1] veröffentlicht. Die Definition der Schnittstelle *IAdapter* zeigt nachstehende Abbildung 5-26. Jeder für den **dbEngineer** erstellte Adapter muss sich an diese Definition bindend halten.

```
 [
  uuid(CB467140-84F5-4B11-94D2-41A6E2C2AC4D),
  version(1.0),
  helpstring("Adapter Bibliothek")

]
library Adapter
{

  importlib("stdole2.tlb");

 [
    uuid(011780DC-AEEA-4EE5-8C26-8996FD5418B9),
    version(1.0),
    helpstring("Dispatch-Schnittstelle für Adapter-Objekt"),
    dual,
    oleautomation
  ]
   interface IAdapter: IDispatch
  {
   [
    id(0x00000001)
    ]
    HRESULT _stdcall Query_ID( [out] VARIANT * VerStr, [out] VARIANT * Liste );
   [
    id(0x00000002)
    ]
    HRESULT _stdcall StartGen( [in] long ID, [in] BSTR ProjInfo, [in] BSTR Ortaspek );
   [
    id(0x00000003)
    ]
    HRESULT _stdcall Generate( [in] long ID, [in] BSTR Aktor, [in] BSTR Modul );
   [
    id(0x00000004)
    ]
    HRESULT _stdcall EndGen( [in] long ID );
  };

 [
    uuid(FAFBFE30-794A-4ADA-AC08-020F1A9B2673),
    version(1.0),
    helpstring("Ereignisschnittstelle für Adapter-Objekt")
  ]
   dispinterface IAdapterEvents
  {
    properties:
    methods:
   [
    id(0x00000001)
    ]
    HRESULT OnAfterStartGen( [in] long Success, [in] BSTR Info );
   [
    id(0x00000002)
    ]
    HRESULT OnBeforeGenerate( [in] long Success, [in] BSTR Info );
   [
```

[1] IDL Abk. für: **I**nterface **D**efinition **L**anguage (engl.): Schnittstellen-Beschreibungssprache

```
    id(0x00000003)
    ]
    HRESULT OnAfterGenerate( [in] long Sucess, [in] BSTR Info, [in] BSTR Aktor);
   [
    id(0x00000004)
    ]
    HRESULT OnBeforeEnd( [in] long Success, [in] BSTR Info );
   [
    id(0x00000005)
    ]
    HRESULT OnAfterEnd( [in] long Sucess, [in] BSTR Info, [in] BSTR Ortsaspekt);
  };

 [
    uuid(B0993036-2AB2-4BB5-9FF9-838DAACA9BFA),
    version(1.0),
    helpstring("Adapter Objekt"),
    custom(B0FC9342-5F0E-11D3-A3B9-00C04F79AD3A, True),
    custom(B0FC9344-5F0E-11D3-A3B9-00C04F79AD3A, 3),
    custom(17093CC8-9BD2-11CF-AA4F-304BF89C0001, 0),
    custom(B0FC9343-5F0E-11D3-A3B9-00C04F79AD3A, True)
  ]
  coclass Adapter
  {
   [default] interface IAdapter;
   [default, source] dispinterface IAdapterEvents;
  };

};
```

Abbildung 5-26: Definition für das Interface *IAdapter* als IDL

Die gesamte Funktionalität der Adapter wird definitionsgemäß über vier Methoden abgewickelt:

- *Query_ID*: Mit dem Aufruf dieser Methode kann der **dbEngineer** die unterstützten Methoden des installierten Adapters abfragen. Der Adapter liefert eine Auflistung aller unterstützten Funktionen zurück.
- *StartGen*: Der Adapter kann bei Aufruf dieser Methode vorbereitende Aufgaben intern oder im Zusammenspiel mit dem Planungswerkzeug durchführen, wie beispielsweise das Anlegen von Projektverzeichnissen oder Einrichten von Protokolldateien.
- *Generate*: Mit dieser Methode wird das Erzeugen des Engineerings angestoßen. Diese Methode wird rekursiv für jeden Aktor durchgeführt bis die gesamte Aktorliste durchlaufen wurde.
- *EndGen*: Der Adapter kann bei Aufruf der Methode in Zusammenarbeit mit dem Planungswerkzeug abschließende Aufgaben durchführen, wie beispielsweise temporäre Dateien löschen, Dateien schließen oder Protokolldateien ausgeben.

Damit der Adapter dem **dbEngineer** geänderte Datenmengen und allgemeine Statusmeldungen über den Verlauf des Generiervorgangs übergeben kann, enthält die Definition des Interfaces für den Adapter in Abbildung 5-26 eine Eventschnittstelle *IAdapterEvents*. Mit Hilfe dieser in der

Schnittstelle definierten Events erfolgt an festgelegten Stellen des Funktionsablaufs, den Aktionspunkten, eine Rückmeldung des Adapters an den **dbEngineer**. Das Zusammenspiel zwischen Generiervorgang und den darin eingebetteten Events zeigt das Flussdiagramm in Abbildung 5-27. Der **dbEngineer** legt bei jedem Generiervorgang einen neuen Datensatz in der Tabelle *Logdatei* an, der den Ablauf des Generiervorgangs dokumentiert. Die Dokumentation erfolgt auf Basis der vom Adapter ausgelösten Events und deren Informationsstrings. Am Ende des Generierlaufs kann der Anwender mit Hilfe der *Logdatei* die einzelnen Schritte und deren Meldungen nachvollziehen.

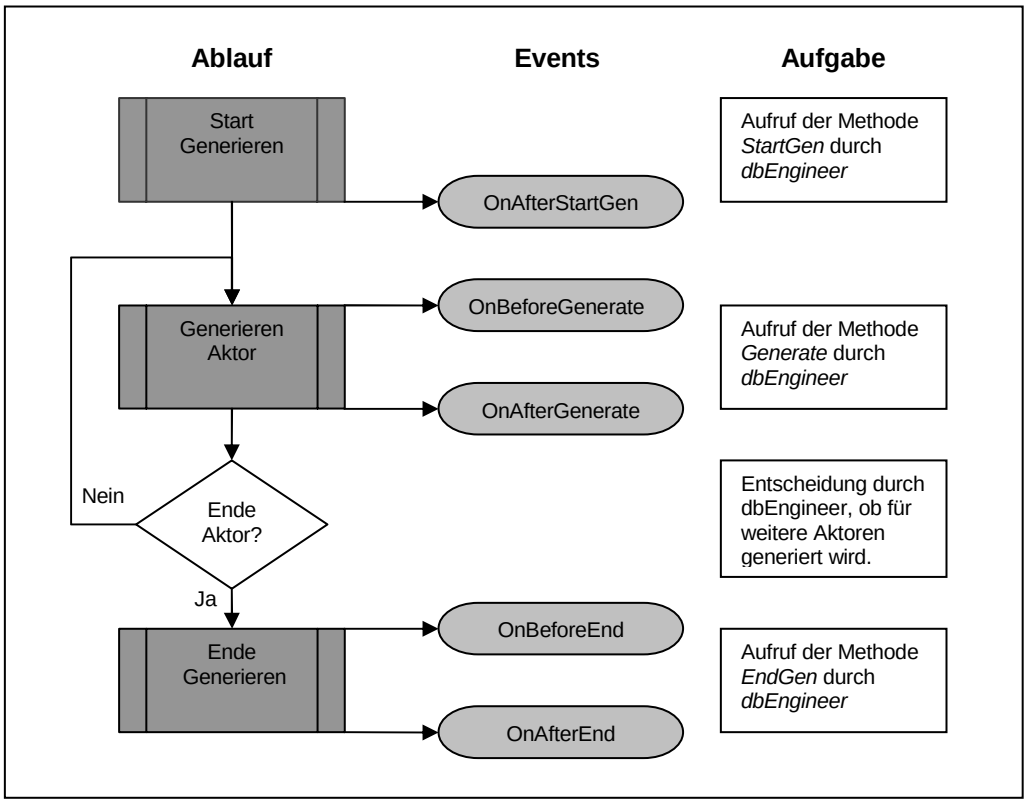

Abbildung 5-27: Ablauf des Generiervorgangs mit Aktionspunkten, den Events des Adapters

Für die Kommunikation zwischen Adapter und **dbEngineer** wurden insgesamt fünf Events definiert. Die Events übergeben dem **dbEngineer** spezifizierte Parameter zur Auswertung. Allen fünf Events gemeinsam sind die Parameter *Success* und *Info*, mit Hilfe dieser Parameter informiert Adapter, ob die gewünschte Aktion erfolgreich war oder fehlschlug. Enthält *Success* eine „1“ war die Aktion erfolgreich, jeder andere Inhalt bedeutet, dass die Aktion fehlschlug. Hat der Adapter bis zum Zeitpunkt des Sendens mehrere interne, dem **dbEngineer** auf Grund der Implementierung nicht bekannte, einzelne Aktionen ausgeführt, darf der Adapter *Success* nur dann auf „1“ setzen, wenn alle Schritte erfolgreich waren. Ist nur ein einziger Schritt im Adapter oder Planungswerkzeug

fehlgeschlagen, muss ein anderer Wert als „1“ gesetzt werden. In der Variablen *Info* übergibt der Adapter weitere Informationen zum Status der Aktion im Klartext. Die Variable *Info* ist vom Typ *BStr*, d.h. sie ist nicht längenbegrenzt. Vom Adapter wird erwartet, dass für jeden einzelnen Schritt, der fehlgeschlagen ist, der String *Info* um dessen Rückmeldung verlängert wird.

Wie der **dbEngineer** im Einzelfall mit der Rückmeldung *Success* umgeht, ist bei den einzelnen Events erläutert.

- *OnAfterStartGen*: Nach Abarbeitung des Aufrufes *StartGen*, wird dieser Event ausgelöst. Der Adapter hat zu diesem Zeitpunkt die übergebenen Datenmenge geprüft und je nach Adaptertyp die Verbindung zu seinem Planungswerkzeug und dem Projekt aufgebaut, oder das Erzeugen einer Steuerdatei vorbereitet. Schlägt eine dieser Aktionen fehl, wird über den Parameter *Success* ein Fehler gemeldet und durch *Info* spezifiziert. Der **dbEngineer** bricht bei einem Fehler die weitere Bearbeitung des Vorgangs ab. Bei fehlerfreiem Durchlauf wird *Success* auf „1“ gesetzt und der Generiervorgang läuft weiter.
- *OnBeforeGenerate*: Der Event wird vom Adapter unmittelbar vor dem Aufruf des Planungswerkzeuges gesendet. Zu diesem Zeitpunkt hat der Adapter die übergebenen Daten geprüft. Im Fehlerfall wird das Planungswerkzeug nicht aufgerufen. Dem **dbEngineer** wird über *Success* und *Info* der Status gesendet. Der **dbEngineer** kann zu diesem Zeitpunkt nicht eingreifen (vgl. Abbildung 5-27), sondern protokolliert nur den Zustand für eine spätere Analyse in der Tabelle *Logdatei* mit.
- *OnAfterGenerate*: Der Aufruf des Events erfolgt nachdem der Adapter das Planungswerkzeug zum Generieren aufgerufen hat und dessen Rückmeldung. Über den Erfolg des Aufrufs wird der **dbEngineer** wie beschrieben durch die Variablen *Success* und *Info* informiert. Handelt sich um einen Vorgang bei dem Adapter oder Planungswerkzeug die Datenmenge *Aktor* verändert haben, werden die Daten durch den Event an den **dbEngineer** zurückgeliefert. Dabei gilt, dass ein leerer String übergeben wird, wenn die Daten nicht verändert wurden. Wird über *Aktor* ein Datensatz zurückgegeben und *Success* führt eine „1“ für fehlerfrei, trägt der **dbEngineer** die Daten über die Methode *UpdateItem* des Servers *Projekt* in die Projektdatenbank ein.
- *OnBeforeEnd*: Der Adapter löst diesen Event aus, wenn das Ende der Generierung im Zusammenspiel mit dem Planungswerkzeug erfolgt ist. Der **dbEngineer** wertet *Success* aus. Wird zu diesem Zeitpunkt ein Fehler gemeldet, gilt der Generiervorgang als fehlge-

schlagen und der **dbEngineer** führt in der Projektdatenbank ein *Rollback*[1] aus, wenn während des Generiervorgangs Datensätze geändert wurden (siehe *OnAfterGenerate*). Ansonsten werden die Änderungen in der Projektdatenbank mit einem *Commit*[2] übernommen.

- *OnAfterEnd*: Nachdem der Adapter seine Verbindung zum Werkzeug und seine abschließenden Vorgänge zum Generieren beendet hat, wird dieser Event ausgelöst.

Auffinden von Adapter und Planungswerkzeug

Es ist typisch für Unternehmen der Automatisierungstechnik mehrere unterschiedliche Planungssysteme für dieselbe Aufgabe zu betreiben und nach Kundenvorgabe ein System für das jeweilige Projekt auszuwählen [3]. Zu den Aufgaben des **dbEngineers** gehört es, das für die Projektierung erforderliche Werkzeug über die Daten der Tabelle *Projektdefinition* zu bestimmen. Weil die Tabelle *Projektdefinition* ein Teil der projektspezifischen Datenbank ist, erhält der **dbEngineer** bei einem Projektwechsel geänderte Daten für die Werkzeuge und deren Standorte. Dadurch wickelt der **dbEngineer** die Planungsunterstützung projektspezifisch mit anderen Werkzeugen ab.

Wie der **dbEngineer** aus den Daten die Standorte (Rechner) von Werkzeug und Adapter ermittelt und sich mit ihnen verbindet, wird im Folgenden dargelegt.

Der Zugriff auf das Planungswerkzeug erfolgt in zwei Stufen, wie Abbildung 5-28 illustriert:

1. *Zugriff auf den Adapter:* Der benötigte Adapter wird aus der Windows Registry[3] ermittelt. Der **dbEngineer** legt in der Registry Informationen (siehe Abbildung 5-29) über alle installierten Adapter ab. Mit den lokal gespeicherten Daten wird der Adapter aufgefunden und angesprochen.

2. *Zugriff auf das Planungssystem*: Der Adapter greift auf das Planungswerkzeug mit den Daten der Tabelle *Projektdefinition* zu. Die Tabelle enthält die erforderlichen Informationen wie beispielsweise Rechnername (IP-Adresse) oder den Projektnamen unter dem die

[1] Rollback (engl.): Zurücknehmen

[2] Commit (engl.): Bestätigung

[3] Windows Registry (engl.): Windows Registrierdatenbank

Projektierung auf dem Planungswerkzeug abgelegt ist. Der Adapter erhält die Daten der Tabelle *Projektdefintion* mit dem Aufruf der Methode *StartGen* als Parameter *ProjInfo*.

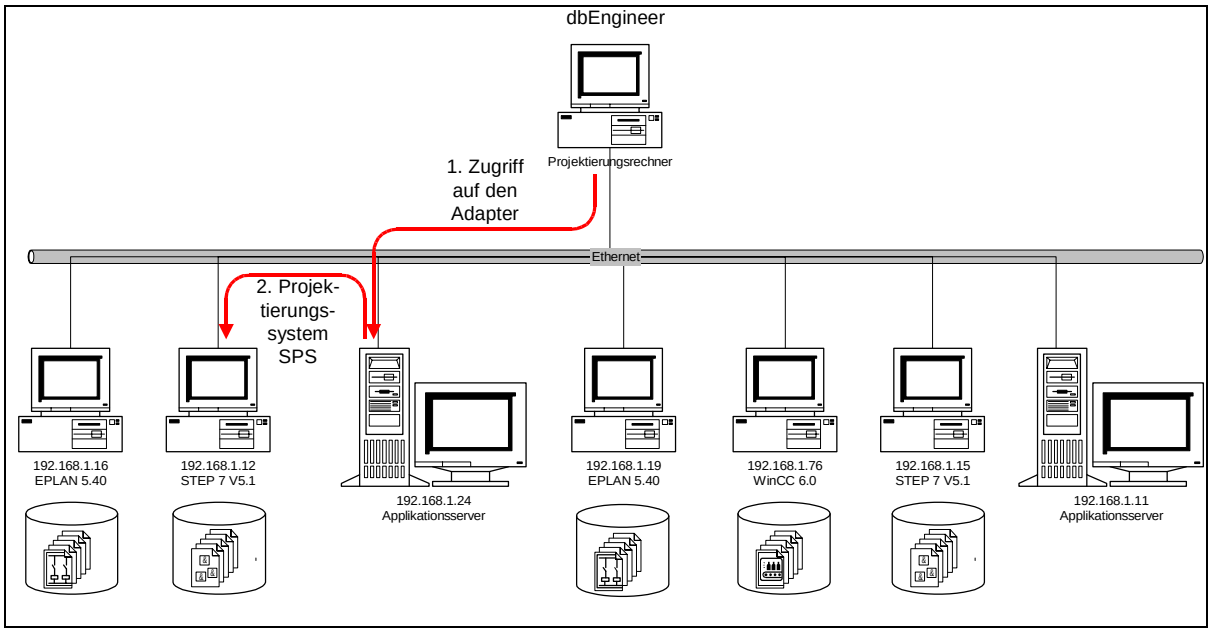

Abbildung 5-28: Auffinden von Applikationsserver (Adapter) und Projektrechner im Netz

Schritt 1: Auffinden von Adaptern

Der **dbEngineer** führt eine lokale Liste gemäß Abbildung 5-29 mit installierten Adaptern auf die der Zugriff möglich ist. Die Liste enthält neben dem Namen des Adapters auch Informationen über den Standort des COM.-Servers in Form der *IP-Adresse* (Rechner) und seiner *ClassID*[1] mit der der Server eindeutig identifiziert werden kann. Über die Liste installierter Adapter kann der Anwender eine beliebige Anzahl von Planungswerkzeugen in Zusammenarbeit mit dem **dbEngineer** nutzen, ohne dass die Programmierung des **dbEngineers** verändert werden muss.

[1] ClassID: Gemäß COM-Spezifikation muss jedes COM-Objekt über eine eindeutige Identifikation verfügen, der ClassID. Es handelt sich dabei um einen GUID.

Add-Ins (Adapter)

Name	Aufrufname	Rechner	ClassID
EPLAN 5.40	eplan540	192.168.1.11	CB467140-84F5-4B11-94D2-41A6E2C2AC4D
STEP 5 V7.2	step5v72	192.168.1.24	B0FC9344-5F0E-11D3-A3B9-00C04F79AD3A
STEP 7 V5.1	step7v51	192.168.1.24	17093CC8-9BD2-11CF-AA4F-304BF89C0001
WinCC V6.0	winccv60	192.168.1.107	FAFBFE30-794A-4ADA-AC08-020F1A9B2673

Abbildung 5-29: Liste installierter Adapter im **dbEngineer**

Wie der **dbEngineer** die erforderliche Komponente als Adapter zu den Entwicklungswerkzeugen findet, wenn für dieselbe Aufgabe mehrere Werkzeuge und damit auch Adapter existieren, zeigt Abbildung 5-30. Ist der Zugriff auf ein externes Werkzeug erforderlich, wird in der Tabelle *Projektdefinition* des Projekts nach dem Werkzeugtyp (E-CAD, SPS oder HMI) gesucht und die Einstellung für das konkrete Werkzeug ausgelesen. Mit dieser Information kann in der Liste der Add-Ins der Adapter zu den Werkzeugen und die zugehörigen Einstellungen gefunden werden. Abbildung 5-30 zeigt beispielhaft die Einstellwerte für den Adapter: den Aufrufnamen für den COM-Server und dessen Standort als IP-Adresse des Hostrechners.

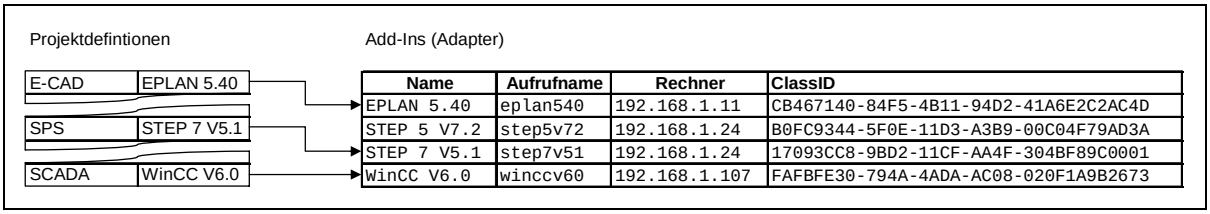

Abbildung 5-30: Auffinden des projektzugehörigen Adapters

Mit den obigen Konfigurationswerten kann der jetzt die Verbindung zum gewünschten COM-Objekt vom Typ *TAdapter* aufgebaut werden. In der Abbildung 5-31 wird eine Verbindung zu einem COM-Server, der sich auf einem entfernten Rechner befindet, erstellt.

```
function (const MachineName: WideString; const ClassID: TGUID) : TAdapter
Var
 V : Variant;

begin
  try
    V := CreateRemoteComObject(MachineName, ClassID);  //Objekt auf entferntem Rechner
    V := unAssigned;                                   // aufrufen und verbinden

    result :=  V as TAdapter; // Typumwandlung
  except
    Result := EmptyParam;     // Fehler beim Aufruf des COM-Servers
  end;
end;
```

Abbildung 5-31: Aufruf eines COM-Objektes auf einem entfernten Rechner (Borland Delphi)

Für den Verbindungsaufbau werden der *Rechnername* bzw. seine *IP-Adresse* (MachineName) und die *ClassID* der Schnittstelle benötigt. Die zurück gelieferte Schnittstelle wird zur weiteren Benutzung in den allgemeinen Typ *TAdapter* umgewandelt, um danach mit den standardisierten Methoden für einen Adapter bearbeitet zu werden. Die Anwendung **dbEngineer** kann mit Hilfe dieser Mechanismen auf die unterschiedlichen Adapter der Geschäftslogik zugreifen, sofern sie vom Typ *TAdapter* sind. Nachdem der **dbEngineer** den Zugriff auf einen Adapter erhalten hat, muss dieser seinerseits auf das zugehörige Planungssystem zugreifen.

Schritt 2: Auffinden von Planungswerkzeugen

Das Planungswerkzeug kann sich auf einem dritten Rechner befinden, dessen Standort (IP-Adresse) der Adapter kennen muss. Gemäß Definition kann jedes Projekt auf einem anderen Planungsrechner ablaufen und der Adapter muss den projektzugehörigen Rechner erreichen können. Der Rechnerstandort gehört deshalb zu den Daten, die in der Tabelle *Projektdefinition* der Projektdatenbank abgelegt sind. Der **dbEngineer** übergibt dem COM-Objekt *TAdapter* mit dem Methodenaufruf *StartGen* im Parameter *ProjInfo* die komplette Tabelle *Projektdefinition* und damit die IP-Adresse für die Verbindung zum Planungswerkzeug.

Die Übergabe der *ClassID* für die Objektschnittstelle des Planungswerkzeugs an den Adapter ist nicht vorgesehen. Der Adapter ist ein spezialisierter COM-Server für den Zugriff auf ein einziges Planungswerkzeug, deshalb wird erwartet, dass dem COM-Server alle Spezifiktionen der Schnittstelle bekannt sind. Die einzig zulässige Variable für den Verbindungsaufbau des Adapters ist der Standort (Rechnername / IP-Adresse) des Planungswerkzeuges.

Für den Adapter gestaltet sich der Zugriff auf seinen zugehörigen Automatisierungsserver (Aufruf des benötigten Planungswerkzeugs) einfacher. Der Adapter verbindet sich nur mit einem bereits bei der Entwicklung des Adapters bekannten Objektmodell eines Planungswerkzeuges. Bei der Entwicklung kann die Typbibliothek des Servers importiert und bei den meisten Programmiersprachen in die Entwicklungsumgebung integriert werden. Ein COM-Objekt hat beispielsweise nach dem Import in Borland Delphis Klassenbibliothek VCL[1] einen Aufbau gemäß folgendem Schema, wie in Abbildung 5-32 gezeigt.

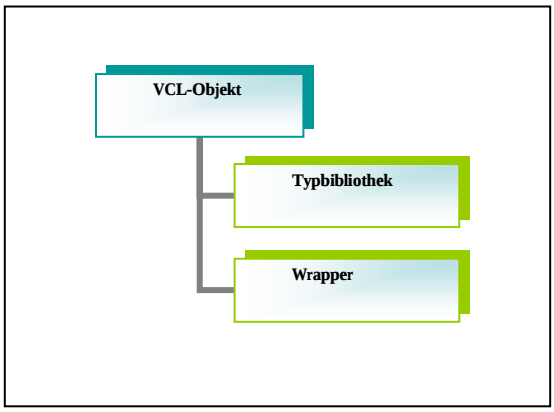

Abbildung 5-32: Aufbau eines eingekapselten COM-Servers in der VCL

Der Komponenten-Wrapper ergänzt die veröffentlichen Methoden und Events der Typbibliothek um Mechanismen zum Verbindungsaufbau, Remotezugriff und Rechnernamen. Der programmtechnische Zugriff auf den Automatisierungsserver vereinfacht sich dadurch erheblich, weil z.B. die *ClassID* des COM-Servers bereits durch den Wrapper importiert ist oder die Abfragen über das Interface *IUnknown* entfallen. Den Zugriff auf einen Automatisierungsserver, hier auf das SPS-Programmierwerkzeug STEP 7, zeigt das Codefragment einer Borland Delphi Programmierung in Abbildung 5-33.

[1] VCL Abk. für **V**isual **C**omponent **L**ibrary (engl.): Visuelle Komponentenbibliothek; Klassenbibliothek für Borland Programmiersprachen Delphi und C++.

```
...

Var
  S7 : TAdapt_S7;          // S7 als Variable vom COM-Servertyp S7 deklarieren
...
...

  S7.MachineName := Rechnername;   // IP-Adresse über Rechnername zuweisen
  S7.Connect;                      // Mit dem COM-Server S7 verbinden

...
```

Abbildung 5-33: Aufruf eines eingekapselten COM-Servers (Borland Delphi)

Nachdem die einzelnen Schritte der Zugriffe auf Adapter und Softwarewerkzeug erläutert wurden, soll das gesamte Zusammenspiel von **dbEngineer**, Adapter, Werkzeug, der lokalen Installationsliste für Adapter und der Tabelle *Projektdefinition* vorgestellt werden. Dazu dient das Beispiel der Projekte A und B gemäß Abbildung 5-34, mit dem nachvollzogen werden kann, wie das Auffinden der einzelnen Komponenten in einem Netzwerk erfolgt.

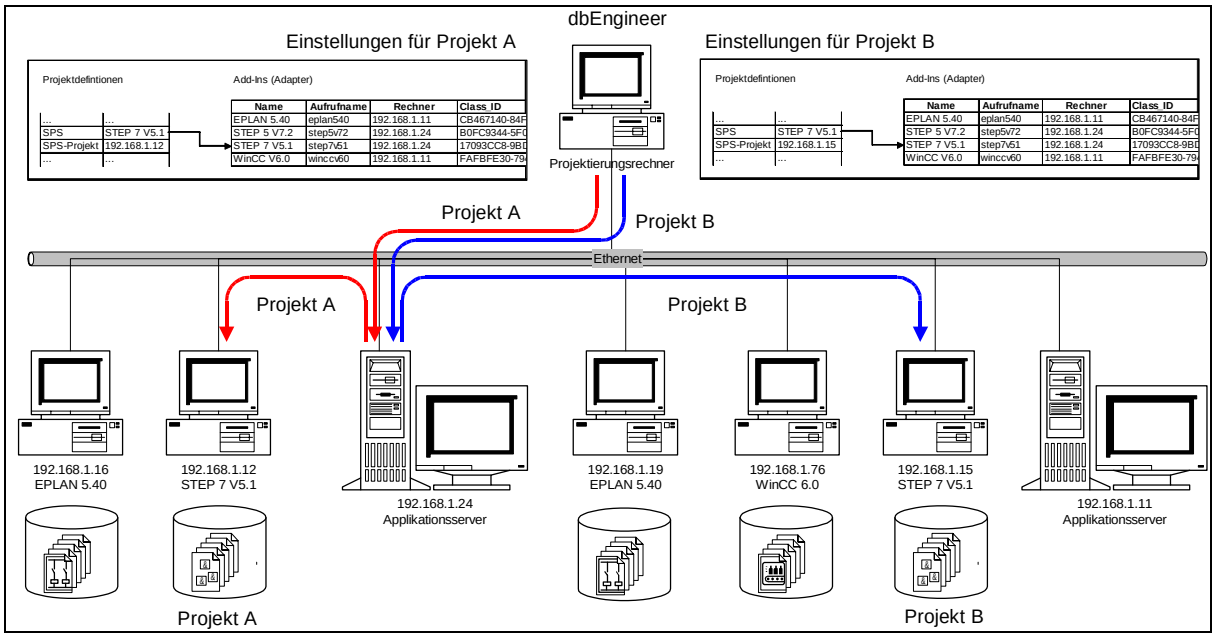

Abbildung 5-34: Auffinden von Applikationsserver (Adapter) und Projektrechner im Netz

Mit der Aufteilung des **dbEngineers** in eine mehrschichtige Architektur und den damit verbundenen Vorteilen, wie der dynamischen Verwaltung von Standorten der einzelnen Schichten der Anwendung oder dem Zufügen von Adaptern, ist es möglich, die vom **dbEngineer** erwartete Flexibilität, Skalierbarkeit und Erweiterbarkeit zu gewährleisten.

5.4.2 Die Schnittstelle I_MuK

Die Schnittstelle *I_MuK* wird spezifiziert, damit Dritte einen Import-/Exportfilter für die MuK-Liste erstellen zu können. Standardmäßig unterstützt der **dbEngineer** den Import und Export des definierten XML-Formates der MuK-Liste, wie im 5.3.3 *Datenstruktur für die MuK-Liste* beschrieben. Der standardmäßige Import/Export durch den mitgelieferten Import/Export-Adapter des **dbEngineers** erfolgt über ein physisches Laufwerk des Clientrechners als XML-Datei.

Um die Flexibilität und Erweiterungsfähigkeit des Systems zu garantieren, besteht die Möglichkeit weitere COM-Server einzubinden. Die Mechanismen der Einbindung und deren Verwaltung entsprechen denen der Adapter zu den Anwendungen. Beispiele für Erweiterungen sind zusätzliche Import/Exportfilter (COM-Server), die durch den Anwender oder durch Dritte erstellt werden können:

- Import/Export von und zu weiteren Datenformaten z.B. aus der Officewelt
- Direkter Zugriff auf eine Datenbank, in der die MuK-Liste abgelegt ist
- Lesen und Schreiben der MuK-Liste von einem WebServer über das Internet/Intranet.

Der **dbEngineer** erwartet, dass der Server, der den Import oder Export vornimmt, alle Vorgänge, die im Server ablaufen, gegenüber der Anwendung abstrahiert und abschottet. Die Zugriffe von und zur Anwendung werden über die nachstehenden Methodenaufrufe abgewickelt:

- *SetProjekt*: Der COM-Server erhält mit dieser Methode alle Angaben zur Identifikation des Projektes als XML-Datei.
- *IsFileBased*: Mit dieser Methode kann die Anwendung abfragen, ob es sich bei diesem Server um einen dateibasierten Import/Export handelt. Von der Methode wird ein Dateiendungsfilter zurückgeliefert, mit dem die Anwendung die Windowsdialoge *Datei Öffnen/Speichern* vorbelegen kann. Ist der zurück gelieferte String für den Rückgabewert *Leer,* handelt nicht um einen dateibasierten Import/Export. In diesen Fall wird vor dem Import/Export durch die Anwendung kein Windowsdialog zum Öffnen/Speichern von Dateien aufgerufen.
- *ReadMuK*: Mit der Methode wird eine MuK-Liste im XML-Format zurückgeliefert. Die Anwendung erwartet eine MuK-Liste im aufbereiteten internen XML-Format gemäß Definition. Der Methode wird der Standort der MuK-Liste übergeben, wenn es sich um einen Dateizugriff handelt, sonst ist der Übergabeparameter *Standort* leer. Der Im-

port holt die MuK-Liste im nativen Format und übernimmt die Konvertierung in das anwendungsinterne XML-Format.

- *WriteMuK*: Die Methode dient dem Schreiben der MuK-Liste, die auch hier im internen XML-Format übergeben wird. Als zweiter Aufrufparameter wird für dateibasierte Schreibvorgänge das Ziel für den Schreibvorgang übergeben, sonst ist der Übergabeparameter *Standort* leer. Der Export wandelt die MuK-Liste vom XML-Format in das geforderte native Format um.

Die Anwendung **dbEngineer** stellt sicher, dass die Methoden *SetProjekt* und *IsFileBased* aufgerufen werden, bevor die Methoden *ReadMuK* und *WriteMuK* benutzt werden.

Benötigt der installierte Server weitere Informationen vom Anwender, die nicht über die Schnittstelle übergeben werden, weil sie der Anwendung nicht bekannt sind, so wird erwartet, dass der Server einen eigenen Dialog z.B. als ActiveX für die erforderlichen Angaben öffnet. Mit den Informationen des Projektes als eindeutiger Identifikation, kann der Server sich die Angaben aus den Dialogen, falls erforderlich, lokal ablegen und beim nächsten Aufruf wieder verwenden. Die Verwaltung derartiger Einstellwerte ist Aufgabe des jeweiligen Servers.

*5.4.3 Customizing des **dbEngineers***

Der **dbEngineer** kennt, wie bereits beschrieben, im Auslieferungszustand keine Funktionen zur Planungsunterstützung. Die Funktionen sind Bestandteil der Geschäftslogik: den Adaptern. Nach der Installation von Adaptern und der Existenz einer Tabelle *Projektdefinition*, muss der **dbEngineer**, über die Methode *Query_ID*, die bereitgestellten Funktionen ermitteln. Der Anwender erzeugt mit den zurück gelieferten Informationen des Adapters ein projektbezogenes Menü im **dbEngineer**. **DbEngineer** und Adapterfunktionen werden den Erfordernissen des Projektes angepasst (customizing).

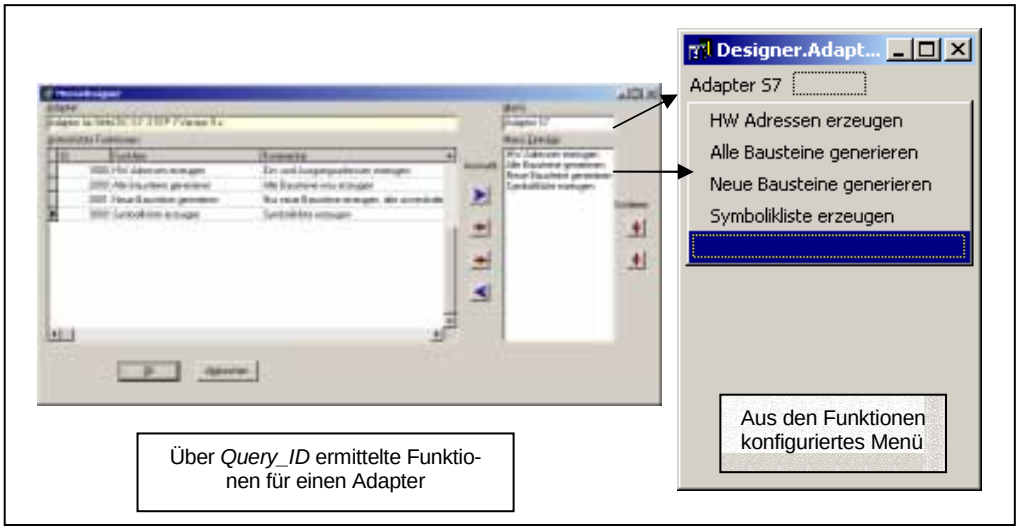

Abbildung 5-35: Customizing des **dbEngineers**

Abbildung 5-35 zeigt diesen Vorgang. Zunächst werden die Funktionen des Adapters ermittelt. Aus den zurück gelieferten Informationen über bereitgestellte Funktionen, selektiert der Anwender die Funktionen, die für das Projekt benötigt werden und fügt sie einem konfigurierbaren Menü im **dbEngineer** zu. Jeder Adapter erhält ein eigenes Menü mit den zugehörigen Menüeinträgen für die selektierten Funktionen des Adapters. Weil in jedem Projekt andere Adapter zusammenarbeiten können, speichert der **dbEngineer** das konfigurierte Menü in der Tabelle *Customizing* der Projektdatenbank. Bei einem Projektwechsel lädt der **dbEngineer** die Konfiguration der Menüs aus der gewechselten Projektdatenbank und baut die Menüleiste projektspezifisch neu auf.

Neuere Entwicklungen zeigen zum Beispiel bei den E-CAD-Systemen, dass diese zu Engineeringwerkzeugen für den kompletten Projektierungsprozess erweitert werden und damit den Anspruch, als E-CAE-Systeme in der Automatisierung eingesetzt zu werden, zum Teil erfüllen [38]. Mit dieser Entwicklung sind umfangreiche Funktionserweiterungen der Systeme verbunden, die über den Umfang der Zeichnungserstellung hinausgehen. Beispielsweise können einige E-CAE Systeme auch Peripherieadressen für bestimmte Fabrikate vom SPS vergeben. Benutzt der Anwender in seinem Projekt ein SPS-Werkzeug, das nicht oder nur bedingt zum Generieren von Peripherieadressen geeignet ist, während aber das Zeichnungssystem die Funktion besser anbietet, ist es sinnvoll, dafür auf das E-CAE System zurückzugreifen und das Menü entsprechend zu konfigurieren.

Mit der Definition eines einzigen Interfaces für alle Adapter wird jedem Planungswerkzeug der gesamte Informationsumfang über einen Aktor übergeben, der im **dbEngineer** abgelegt ist. Mit der Entscheidung nicht für jeden Typ von Planungswerkzeug ein eigenes Interface zu definieren, ist das System flexibel einzusetzen. Jedes Planungssystem erhält alle Daten und könnte damit alle denkbaren Funktionen und nicht nur auf seinen Bereich beschränkt Funktionen ausführen. Durch die Möglichkeit, sich das am besten geeignete Planungssystem für die jeweilige Aufgabe aussuchen zu können, kann sich der Anwender für jedes Projekt und jede Aufgabe die Unterstützung des **dbEngineers** passgenau konfigurieren (Menüeinträge).

Das Customizing ist Aufgabe des Projekteurs und wird nicht einen Administrator übertragen. Mit der Durchführung des Customizings durch den Projekteur steigt die Akzeptanz des Systems, weil er die Fähigkeiten der Planungssysteme und ihrer Adapter im Projektierungsprozess genau kennt und es in seiner Hand liegt, die optimale Auswahl von Funktionen zu bestimmen.

6 EINSATZ DES DBENGINEERS

Die vollständige programmtechnische Umsetzung des vorgestellten Konzeptes **dbEngineer** und seiner Datenstrukturen in ein Softwareprodukt wird mehrere Mannjahre dauern. Um eine genauere Abschätzung des Aufwandes zu ermöglichen, ist die Erstellung eines Lasten- bzw. Pflichtenheftes erforderlich. Die Aufstellung eines Lastenheftes für ein Produkt **dbEngineer** erfordert die Mitarbeit von Unternehmen, die mit einem Prototypen des Systems arbeiten und ihre Erfahrungen und Wünsche, insbesondere in Bezug auf Bedienerfreundlichkeit, Flexibilität bei Änderungen in der Planung und selbstverständlich dem Anwendernutzen einbringen. Für die Einführung eines solchen neuen Systems hat sich daher die *Keimzellenstrategie* bewährt. Bei der Keimzellenstrategie werden die Phasen der Einführung zunächst für einen eingeschränkten Einsatzbereich durchlaufen und nach erfolgreichem Einsatz schrittweise erweitert [18].

Im Sinne dieser Strategie wird der **dbEngineer** zunächst nur mit einem einzigen Adapter zur Kopplung mit einem SPS-Programmiersystem zu Einsatz gebracht. Es soll noch einmal verdeutlicht werden, dass es sich hierbei nur um einen Prototypen zur Verifikation des Konzeptes und nicht um ein Produkt mit komplettem Funktionsumfang handelt.

Angewendet wird eine Vorstufe des Systems von einem mittelständischen Automatisierungsunternehmen bei der Automatisierung einer Siloanlage (Bunkeranlage) und deren Fördertechnik.

6.1 Die Einsatzumgebung

Bevor der Einsatz des **dbEngineers** gezeigt wird, sollen das beteiligte Unternehmen, die Anlage und das Programmiersystem vorgestellt und deren Auswahl begründet werden.

6.1.1 Das Automatisierungsunternehmen

Bei dem Unternehmen, das für einen Einsatz des **dbEngineers** gewonnen werden konnte, handelt es sich um ein mittelständisches Unternehmen aus dem Ruhrgebiet, das mit rund 120 Mitarbeitern/-innen Automatisierungslösungen für Hütten- und Walztechnik, Kläranlagen, Fördertechnik und Chemische Industrie liefert. Dabei werden die Bereiche E-CAD, SPS-Programmierung, Leitsystemerstellung und Betriebsführungssysteme abgedeckt.

Das Unternehmen betreibt ein lokales Firmennetzwerk (LAN[1]) auf Basis des Betriebssystems Windows 2000 Server mit einem DBMS der Firma Oracle für die Unternehmensdaten. Bei der Einführung des **dbEngineers** werden LAN und DBMS genutzt.

6.1.2 Die Anlage

Bei der ausgewählten Anlage handelt es sich um eine Siloanlage für Schüttgüter. Eine Bunkeranlage bietet sich für die Erprobungsphase eines neuen Systems deshalb an, weil sie zwar eine Vielzahl von Aktoren und Sensoren besitzt, diese aber eine begrenzte Variationsbreite von elektro-mechanischen Grundsystemen bilden. Die Anzahl der erforderlichen Module, die für den Testlauf des **dbEngineers** zu erstellen sind, bleibt somit überschaubar. Die Anlage ist aus neun verschiedenen Systemen aufgebaut, wobei rund 80% aller auftretenden elektro-mechanischen Systeme auf nur zwei Varianten zurückzuführen sind. Abbildung 6-1 kann das durch einen Ausschnitt der Anlage als Produktaspekt im **dbEngineer** verdeutlichen.

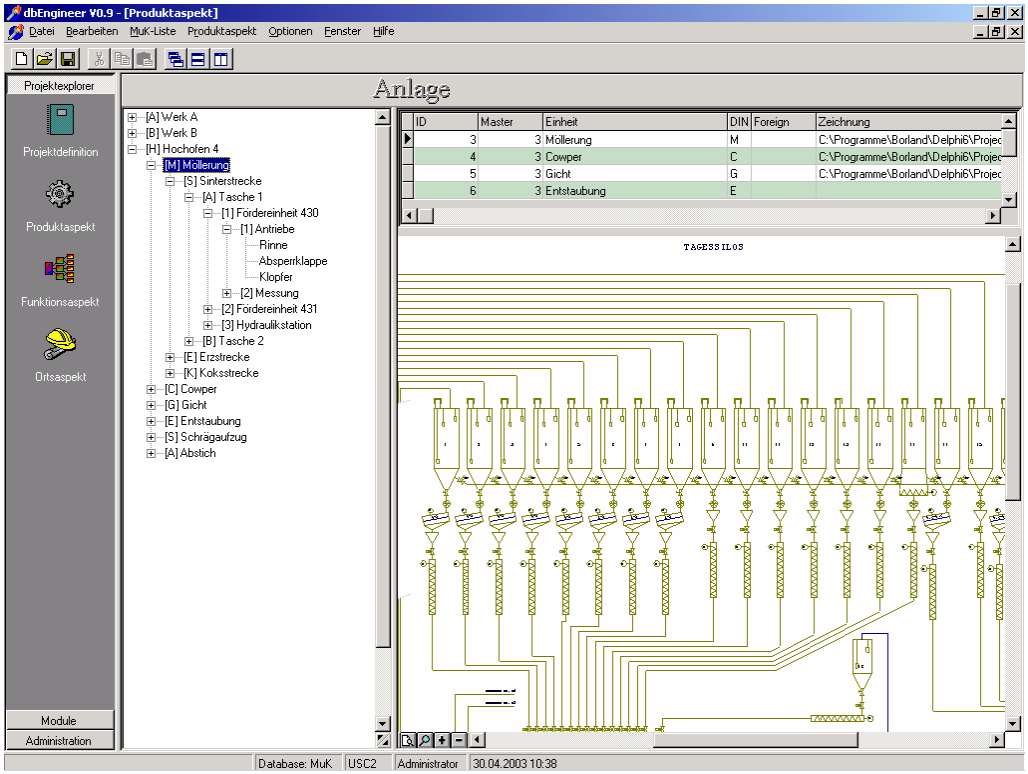

Abbildung 6-1: Silo- oder Bunkeranlage (Ausschnitt) als Produktaspekt im **dbEngineer**

[1] LAN Abk. für **L**ocal **A**rea **N**etwork (engl.): lokales Netzwerk

6.1.3 Das Planungswerkzeug

Als Planungswerkzeug, dessen Ankopplung zu realisieren ist, wurde das SPS-Programmiersystem SIMATIC STEP 7 gewählt. Für die Wahl von STEP 7 sprechen mehrere Gründe:

- die weite Verbreitung als Marktführer bei SPS [12]
- die Veröffentlichung der COM-basierten Objekthierarchie von STEP 7, mit Beschreibung und Programmierbeispielen als Bestandteil der Softwaredokumentation.
- die Möglichkeit Softwarebausteine von STEP 7 als Textdateien zu exportieren und damit in die Modulbibliothek des **dbEngineers** überführen zu können.

Der **dbEngineer** kann mit STEP 7 von der Multi-Tier-Architektur und COM Gebrauch machen.

6.1.4 Die Testumgebung

Bei der Einführung des **dbEngineers** stand die Netzwerk- und Rechnerkonstellation gemäß Abbildung 6-2 zur Verfügung. Für die Tests mit dem **dbEngineer** wurde das Firmennetzwerk um einen zusätzlichen Server 192.168.1.24 erweitert. Der Server diente gleichzeitig als Datenbankserver für die Projekt- und Moduldatenbank, sowie als Applikationsserver für die Module des Datenbankzugriffs und den Adapter zu STEP 7. Die Präsentationsschicht wurde auf einem eigenen Client, dem Projektierungsrechner, eingesetzt.

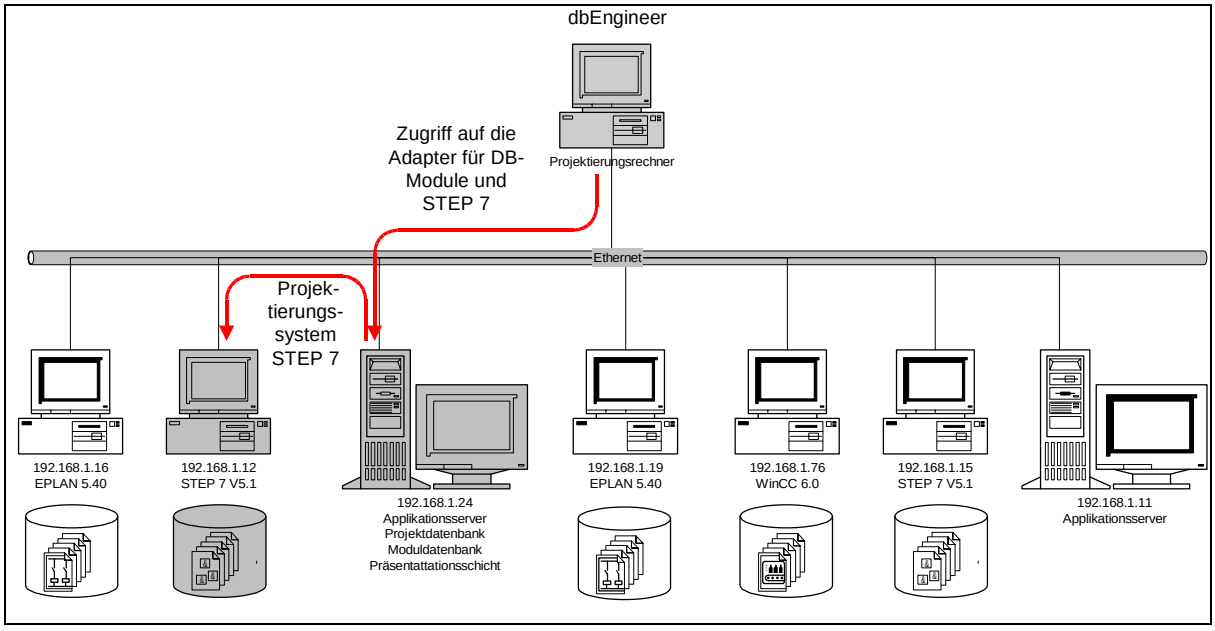

Abbildung 6-2: Die Netzwerkumgebung bei der Einführung des **dbEngineers** (Ausschnitt des LAN)

Mit der verteilten Installation des **dbEngineers** auf zwei Rechnern konnte einerseits die Skalierbarkeit und Flexibilität des Systems gezeigt und andererseits die betriebsbedingten Störungen im Firmennetzwerk durch ein neues System gering gehalten werden, weil durch einfaches Zufügen des Applikationsservers in das Netz die Funktionalität des **dbEngineers** bereitgestellt wurden. Der Server hätte als Präsentationsschicht für die Projektierung genutzt werden können, jedoch sollte die Möglichkeit einer verteilten Anwendung demonstriert werden, daher wurde die Projektierung über einen Clientrechner abgewickelt.

Daten der Testumgebung:

- *LAN*: Windowsbasiert; Windows 2000 Server
- *DBMS*: Oracle 8i mit nativem OLE-DB-Provider
- *Applikationsserver*: Windows 2000 Server
- *Client*: Windows 2000 Professional SP2
- *Projektierungssystem für STEP 7*:
  - Windows 2000 Professional SP2
  - SIMATIC STEP 7 Version 5.1 SP3

6.2 Die Einführungsphase

Die Einführungsphase soll beispielhaft an einigen Bildschirmausschnitten des **dbEngineers** erläutert werden. Die Darstellung des Einsatzes orientiert sich am chronologischen Ablauf der Projektierungsschritte der Automatisierungsplanung.

Zunächst wird das Spezifizieren der Projektdefinitionen dargestellt, gefolgt von der Möglichkeit die Aspektsichten nach DIN EN 61 346 aufzurufen und schließlich wird für einen Aktor exemplarisch der Durchlauf vom Modul in der Modulbibliothek zum Softwarebaustein in STEP 7 gezeigt.

6.2.1 Projektdefinitionen

Zu den grundlegenden Tätigkeiten der Projektarbeit gehört die Projektdefinition. Bei der Definition werden externe Vereinbarungen mit Investor und Anlagenbau getroffen, wie beispielsweise

- Fabrikat und Typ der Planungswerkzeuge: E-CAD, SPS-System und HMI-System oder
- Spannungsebenen für Motore

sowie interne Vereinbarungen, die nur das Automatisierungsunternehmen betreffen, wie beispielsweise

- Projektname in den Planungswerkzeugen oder
- Ablageort der Projektdateien von Planungswerkzeugen.

Diese Informationen legt der **dbEngineer** in seiner Tabelle *Projektdefinition* ab. Sie stehen damit für die weitere Bearbeitung durch den **dbEngineer** und seiner nachgeschalteten Adapter bereit.

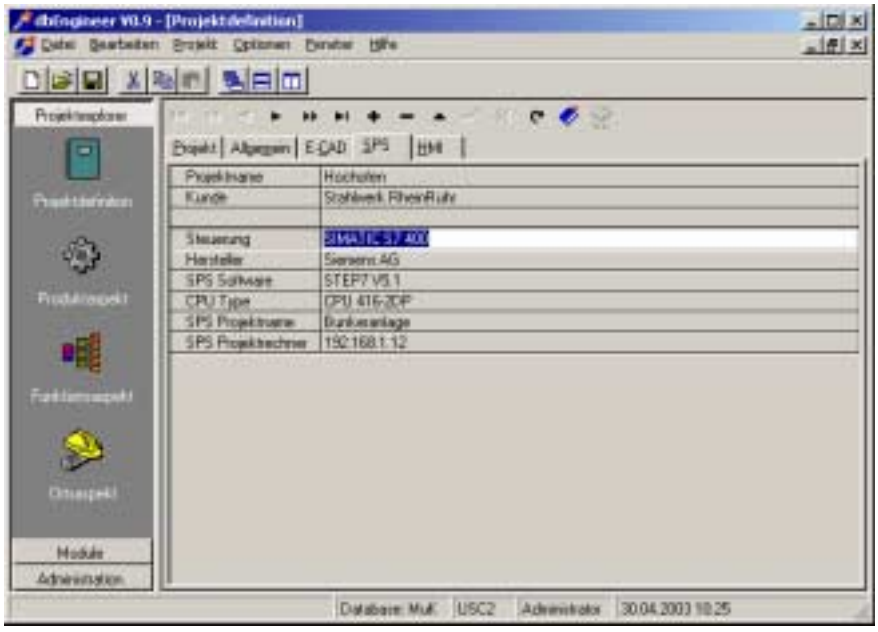

Abbildung 6-3: Tabelle *Projektdefinition*, hier das Register für die SPS im **dbEngineer**

Der **dbEngineer** bietet zur Verwaltung und Eingabe der Projektdefinitionen eine eigene Bildschirmmaske, wie Abbildung 6-3 zeigt, mit Registern für die verschiedenen Werkzeuge an. Die dargestellten Daten dienen dem Zugriff auf den Projektierungsrechner für STEP 7 gemäß Abbildung 6-2. Des Weiteren wird der Suchbaum für das Auffinden von Modulen über das Feld *SPS Software* festgelegt, wie der Ablauf der Projektierung zeigen wird.

6.2.2 Aspekte nach DIN EN 61 346

Nachdem die MuK-Liste in die Tabellen *Aktor* und *Zubehör* überführt wurde, können die Referenzkennzeichen nach DIN vergeben und im **dbEngineer** als Aspekt betrachtet werden. Gemäß den Anforderungen der Norm kann für die Aspekte zwischen den Sichten umgeschaltet werden. Die

Tiefe der Gliederung in den Aspektesichten beträgt sechs Ebenen, gemäß Standardvoreinstellung des **dbEngineers**.

Abbildung 6-4 zeigt den Getriebemotor für eine Rinne als Produktaspekt, dieser Motor ist auf der untersten Stufe der objektorientierten Teilehierarchie angesiedelt. Diese Stufe wird erreicht, wenn ausgehend von der Anlage als der höchsten Stufe, vergleiche Abbildung 6-1, die Baumstruktur des Produktaspekts durchnavigiert wird. Zur Gegenüberstellung zeigt Abbildung 6-5 dieselbe Rinne als Funktionsaspekt.

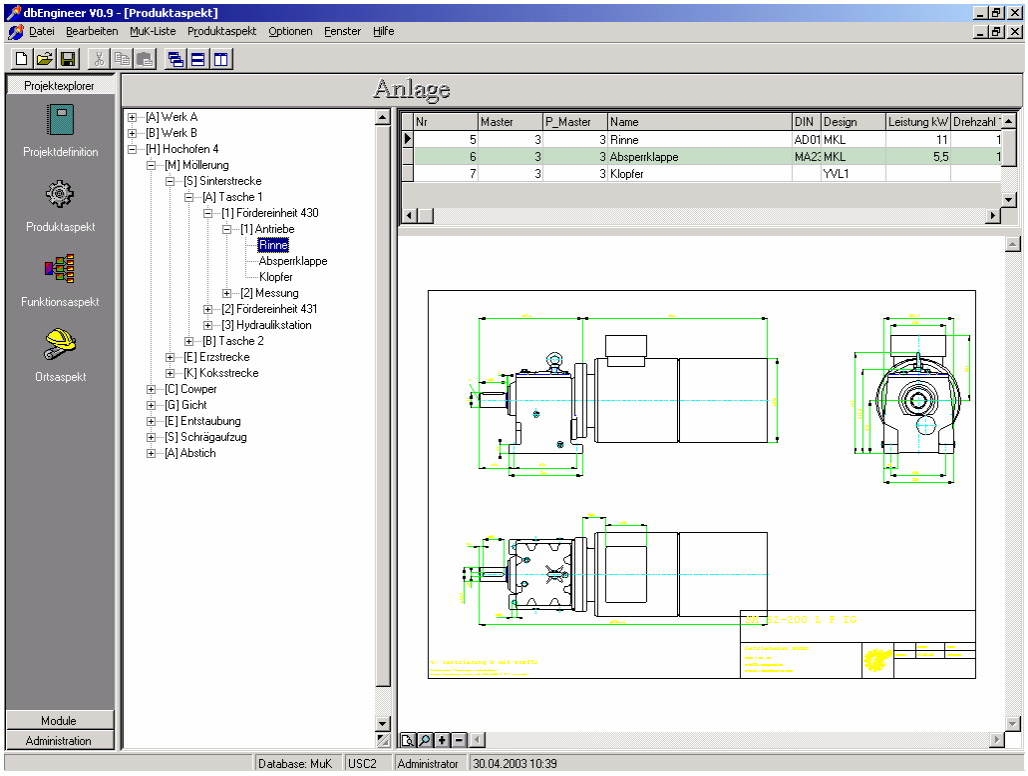

Abbildung 6-4: Getriebemotor für eine Rinne als Produktaspekt

Der Funktionsaspekt zeigt die technischen Daten der Rinne, wie sie aus der MuK-Liste übernommen wurden und die zusätzlich projektierten Daten für die SPS wie beispielsweise:

- *SPS Baustein*: Name unter dem das generierte Programmteil in STEP 7 abgelegt werden soll.
- *SPS AUS 1*: Der SPS-Hardwareausgang zur Schützansteuerung in Richtung 1.
- *SPS AUS 2*: dito für Richtung 2. Da hier ein Antrieb mit einer Drehrichtung vom Typ *KA* vorliegt, bleibt das Feld leer, weil es nicht benötigt wird.

Diese Informationen dienen zur Substitution der Platzhalter im Modul beim späteren Generierlauf. Das benötigte Modul wird vom **dbEngineer** über das Feld *SPS Template* angegeben. Das Feld *SPS Template* ist nicht editierbar und deshalb grau hinterlegt.

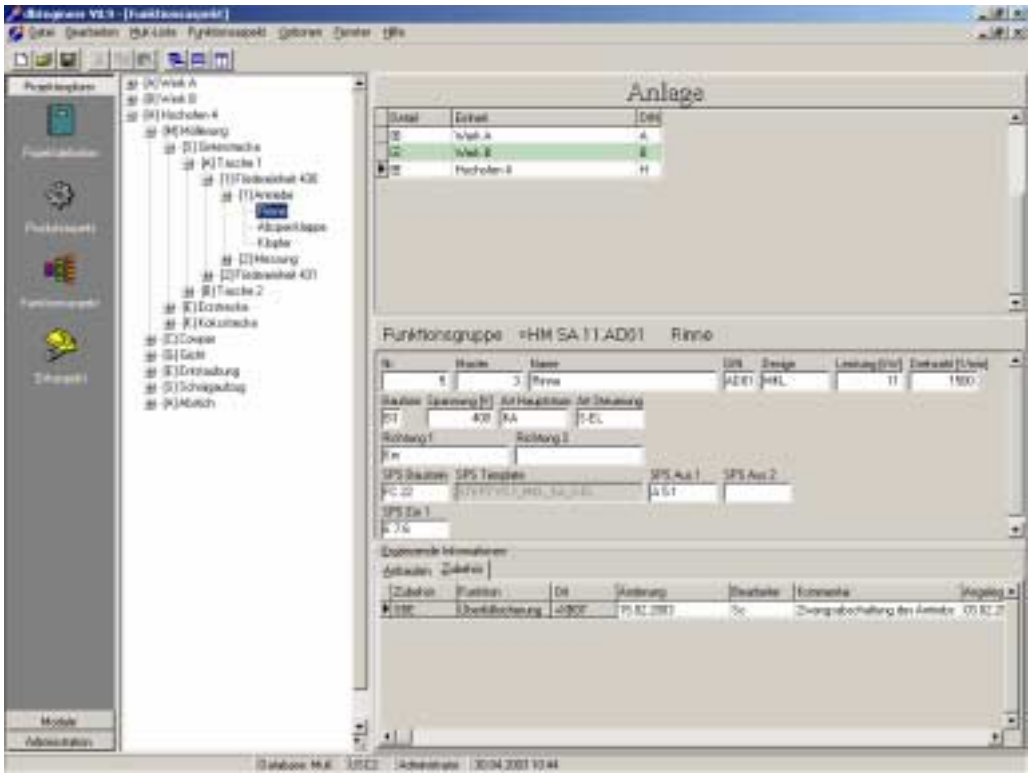

Abbildung 6-5: Getriebemotor für eine Rinne als Funktionsaspekt

In der Baumstruktur des Funktionsaspekts nach Abbildung 6-5 kann die die Bildung des Referenzkennzeichens *=HM SA 11* für das Objekt Rinne nachvollzogen werden. Das Kennzeichen nach dem Gliederungszeichen *AD01* ist ein Feld der Tabelle *Aktor*.

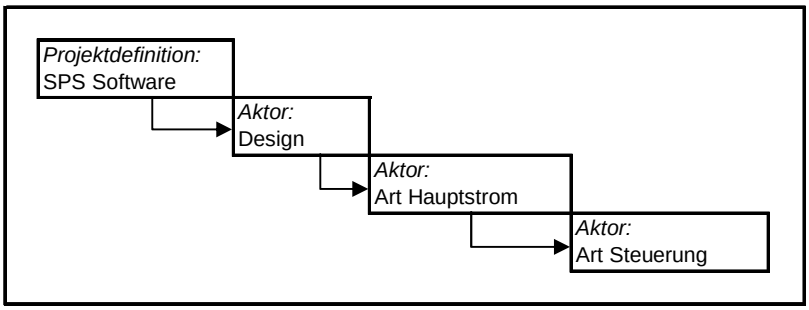

Abbildung 6-6: Vorschrift zur Bildung des Suchschlüssels von SPS-Modulen in der Notation: *Tabellenname:*Feldname

Der Suchschlüssel für das Modul wird vom **dbEngineer** gemäß Abbildung 6-6 ermittelt und in das Feld *SPS Template* der Tabelle *Aktor* eingetragen. Die Bildung des Schlüssels erfolgt über die angegebenen Datenfelder der Tabellen *Projektdefinition* und *Aktor*.

6.2.3 Anbindung STEP 7

Im Rahmen dieser Arbeit wurde die Forderung aufgestellt, dass der **dbEngineer** keine Annahmen über den internen Aufbau und die Arbeitsweise eines Softwareadapters treffen darf. Später wurde ein Konzept vorgestellt, das diese Forderung erfüllt. Es wäre aber nicht sinnvoll, aus dieser Forderung abzuleiten, auch hier beim Einsatz des **dbEngineers** einen Softwareadapter grundsätzlich als black-box[1] zu verwenden. Hier bietet sich die Möglichkeit auch für die Adapterseite aufzuzeigen, dass sich das gewählte Konzept mit der Trennung von Präsentationsschicht und Adapter bewährt. Deshalb wird exemplarisch an Hand von Programmfragmenten die Arbeitsweise des benutzten Adapters vorgestellt.

STEP 7 als windowsbasiertes Programm, mit einer veröffentlichten und gut dokumentierten COM-basierten Schnittstelle für Fremdzugriffe, ist für die Anbindung an den **dbEngineer** über einen Softwareadapter hervorragend geeignet [26]. Die Verbindung zwischen **dbEngineer** und STEP 7 übernimmt der Adapter *TAdapt_S7*. Es wird das Erzeugen eines STEP 7-Programmbausteins FB[2]33 für die Steuerung, der bereits vorgestellten Rinne gezeigt. Die einzelnen Schritte bei der Erzeugung werden an Hand von Bildschirmmasken des **dbEngineers** und Codefragmenten des Adapters dargestellt.

Zunächst soll kurz das Objektmodell von STEP 7 vorgestellt werden auf das der Adapter, der zwischen **dbEngineer** und STEP 7 liegt, zugreift. Für die Funktionsweise und die Handhabung der Programmiersoftware STEP 7 sei auf die mitgelieferte Standarddokumentation bzw. Standardliteratur verwiesen [17] [26].

---

[1] black-box (engl.): hier: Struktur, deren innerer Aufbau nicht offen gelegt wird.

[2] FB Abk. für **F**unktions**b**austein

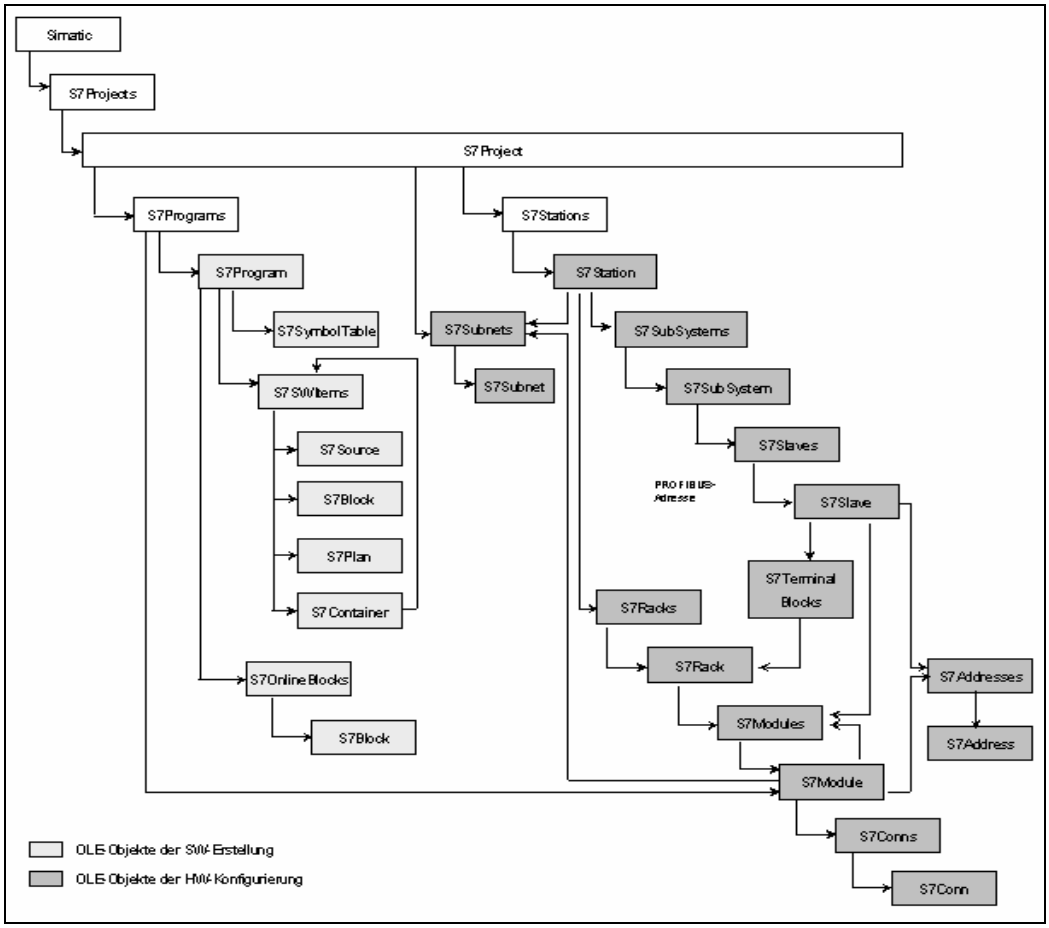

Abbildung 6-7: Objektmodell von SIMATIC STEP 7 Version 5.1 [26]

Das Objektmodell von STEP 7 gemäß Abbildung 6-7 veröffentlicht Klassen für die Softwareerstellung (SW-Erstellung) und die Hardwareerstellung (HW-Konfigurierung). Für den Einsatz des **dbEngineers** werden die Objekte der SW-Erstellung, sowie die weiß hinterlegten Objekte der Projektverwaltung genutzt. Eine Anwahl auf das Projekt muss bei STEP 7 immer über den in STEP 7 vergebenen *Projektnamen* erfolgen.

Nachdem die Verbindung zum Projekt über die Klasse *S7Projects* hergestellt ist, kann der Objekthierarchie folgend auf den Quellcode von Programmbausteinen mit der Klasse *S7Source* zugegriffen werden. Die Klasse *S7Container* verfügt über allgemeine Methoden zur Verwaltung von Quelltexten in STEP 7. Für das Kompilieren des Quellcode ist Klasse *S7Source* zuständig. Sie kann mit ihrer Methode *Compile* entweder alle in *S7Source* abgelegten Quellen oder selektiv einzelne Quellen in Bausteine übersetzen. Kompilierte Bausteine werden von STEP 7 intern separat abgelegt und

über die Klasse *S7Blocks* verwaltet. Für weitergehende Erläuterungen des Objektmodells sei auf die Dokumentation zur Schnittstelle, die der Hersteller veröffentlicht hat, verwiesen [26].

6.2.4 Verwaltung von Funktionsmakros

Grundlage für das automatische Generieren ist die Modulbibliothek. Mit STEP 7 können Musterlösungen erarbeitet werden, die als Quelltext einer AWL[1]-Quelle von STEP 7 exportiert werden können. Der Quelltext ist vollständig textbasiert, er enthält keine Steuerzeichen und kann mit jedem beliebigen Texteditor bearbeitet werden. Auf diese Weise erstellte Quellen können vom **dbEngineer** importiert und verwaltet werden. Die Ablage einer Quelle erfolgt in der Tabelle *Modul* im Feld *Makro*, das Textinformationen beliebiger Länge aufnehmen kann.

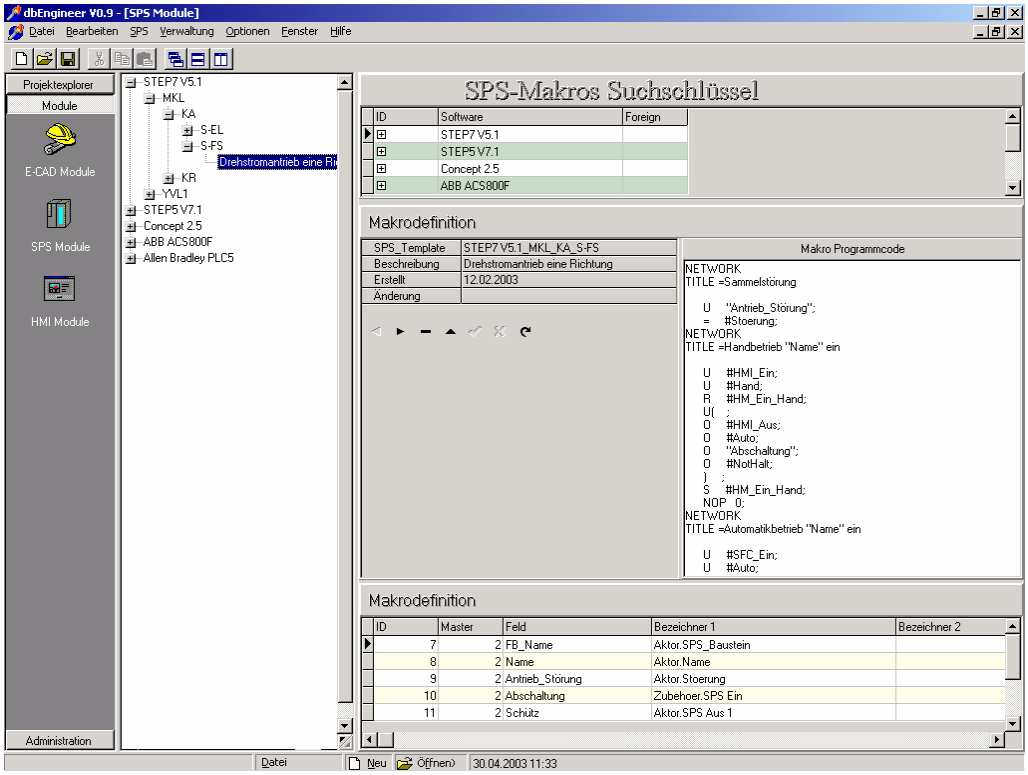

Abbildung 6-8: Makroverwaltung

[1] AWL Abk. für **An**weisungs**l**iste. Eine der drei Sprachen Anweisungsliste (AWL), Kontaktplan (KOP) und Funktionsplan(FUP) aus dem Grundpaket SIMATIC STEP 7. Mit der Installation weiterer SIMATIC-Sprachen werden zusätzliche Quelltypen bereitgestellt, die ebenso behandelt können. Die Arbeit beschränkt sich auf den Einsatz des Grundpakets.

Die Ablage eines Moduls in der Modulbibliothek zeigt Abbildung 6-8. Das Modul ist im Suchbaum dargestellt und besitzt einen eindeutigen Suchschlüssel im Feld *Template*.

Das Modul enthält Platzhalter (Alias) an den Stellen, an denen später die Variablen der Projektierung eingesetzt werden sollen. Die Tabelle *Aliase* speichert die Liste der vorzunehmen Substitutionen. Ihre Spalte *Alias* enthält den Namen des Platzhalters. Die Spalte *Bezeichner* kann entweder auf einen Feldnamen der Tabellen *Aktor*, *Zubehör* oder *Projektdefinition* verweisen oder einen durch Anführungszeichen (″ ″) eingeschlossenen freien Text enthalten. Wird das Modul durch den Adapter bearbeitet (Methode *StartGen*), erfolgt im Quelltext des Moduls eine Substitution der Platzhalter durch den Inhalt der angegebenen Felder oder den Textstring.

6.2.5 Generisches Erzeugen eines Programmbausteins

Nachdem die Objekthierarchie von STEP 7 vorgestellt und die Verarbeitung der Makros erläutert wurde, wird die exemplarische Erzeugung eines Programmbausteins gezeigt. Erzeugt werden soll ein Baustein vom Typ *STEP7V5.1_MKL_KA_S-FS*. Laut Definition handelt es sich dabei um einen Drehstromantrieb mit einer Drehrichtung und Anbindung an eine Ablauf- oder Folgesteuerung, nach Tabelle 6-1.

Kurz	Bedeutung
STEP7V5.1	Baustein für SPS-System: STEP 7 Version 5.1
MKL	Kurzschlußläufermotor für Drehstrom
KA	Eine Drehrichtung
S-FS	Einschalten über Leitsystem (Hand), bzw. **F**olge**s**teuerung (Automatik)

Tabelle 6-1: Bedeutung des Suchschlüssels (Typ) nach MuK-Liste

Bisher wurde der Ablauf eines Generiervorgangs ausschließlich aus Sicht der Präsentationsschicht betrachtet und dargestellt, vgl. hierzu Abbildung 5-27. Im Folgenden werden auch die Abläufe im Adapter berücksichtigt. Der vorgestellte Adapter *TAdapt_S7* wurde mit Borland Delphi 6.02 entwickelt, deshalb entsprechen Syntax und verwendete Objekte in den Codefragmenten dem Programmiersystem ObjectPascal und der Klassenbibliothek VCL.

Start des Generiervorgangs

Der Beginn eines Generiervorgangs wird vom **dbEngineer** mit dem Aufruf der Methode *StartGen* im Adapter eingeleitet. Bei dem Aufruf werden die Tabellen *Projektdefinition* und *Ortsaspekt* für eine weitere Verwendung im Adapter übergeben.

```
procedure TAdapt_S7.StartGen(ID: Integer; const ProjInfo, Ortaspek: WideString);

Var
Rechner, SPSProj, Name : string;

begin
  ...
  // Globale Informationen als XML in Datenmengen sichern
  CDS_ProjInfo.XMLData := ProjInfo;
  CDS_OrtAspek.XMLData := OrtAspek;
  ...
  ...
  // Rechner- und Projektname aus der Tabelle ProjInfo ermitteln
  Rechner := CDS_projInfo.FieldByName('SPS Projektrechner').AsString;
  SPSProj := CDS_projInfo.FieldByName('SPS Projektname').AsString;
  ...
  ...
  // Mit Rechner verbinden
  Try
    ...
    // Verbinden mit dem Rechner
    Simatic.RemoteMachineName := Rechner;
    Simatic.Connect;
    // Erfolgreich
    ...
  Except
    // Fehlerbehandlung
    ...
  End;

  Try
    ...
    // Testen, ob das Projekt vorhanden ist (durch Abfrage des Projektnamens)
    Name := Simatic.S7Projekts [Projekt].Name;
    // Erfolgreich
    ...
  Except
    // Fehlerbehandlung
    ...
  End;

  ...
end;
```

Abbildung 6-9: Codefragment der Methode *StartGen* des Adapters *TAdapt_S7* (Borland Delphi)

Das in Abbildung 6-9 vorgestellte Codefragment zeigt, wie der Adapter den XML-String *ProjInfo* in eine Datenmenge wandelt, aus der die Informationen *Rechnername* und *Projektname* entnommen werden können. Mit den Informationen wird zuerst die Verbindung zum Rechner durch Aufruf des COM-Servers *Simatic* aufgebaut. Anschließend wird geprüft, ob der Server *Simatic* Zugriff auf das gewünschte Projekt hat.

Meldet die Methode *StartGen* über ihre Events eine erfolgreiche Durchführung ihrer Aktionen, so setzt der **dbEngineer** den Generiervorgang fort.

Generieren eines Bausteins

Der **dbEngineer** erzeugt mit dem Aufruf der Methode *Generate* einen Baustein in STEP 7. Der Methode werden als Übergabeparameter die Tabellen *Aktor* und *Modul* mit ihren 1:N verknüpften Tabellen *Zubehör* und *Aliase* als XML-Dokument übergeben.

```
procedure TAdapt_S7.Generate(ID: Integer; const Aktor, Modul: WideString);


Var
BausteinText : string;

begin
  ...
  CDS_Aktor.XMLData := Aktor;
  CDS_Modul.XMLData := Modul;

  // Name des Bausteins für den Aktor ermitteln
  BausteinText := CDS_Aktor.FieldByName('SPS Baustein').AsString;

  // Befehlsparameter dekodieren
  case ID of
    // Funktion für Befehlparameter "S7Comp" ausführen
  S7Comp: begin
            // Platzhalter ersetzen
            ...
            ...
            // Parametrierte "Quelle" in STEP 7 eintragen
            Simatic.Projects [Projekt].Programs [1]. Next ['Quellen'].
            Next.Add(BausteinText, S7Source_, Quelle);
            ...
            ...
            // Der COM-Server S7Source ist für das Compilieren zuständig
            S7Source.ConnectTo(IS7Source(Simatic.Projects [Projekt].Programs [1].
                            Next ['Quellen'].Next [BausteinText]));
            ...
            ...
            // Die zugefügte Quelle kompilieren
            S7Source.Compile;
            ...
            ...
          end;

    // Funktion für Befehlparameter "S7xxxx" ausführen
   S7xxxx: DoSomething_2;
  else ;
    // Unbekannter Befehlparameter
    ErrorKommando;
  end;
end;
```

Abbildung 6-10: Codefragment der Methode *Generate* des Adapters *TAdapt_S7* (Borland Delphi)

Das in Abbildung 6-10 ausgeführte Codefragment des Adapters zeigt die Methodenaufrufe von STEP 7 zum Eintragen und Kompilieren eines Quelltextes. Das Substituieren der Platzhalter im

Quelltext ist aus Gründen der Übersichtlichkeit nicht dargestellt, zudem sind Suche/Ersetze-Operationen für Strings Bestandteil jeder Programmiersprache.

Der eingesetzte Adapter verfügt über einen Testmodus, mit dessen Hilfe die adapterinternen Vorgänge beobachtet werden können. Im Testmodus verfügt der Adapter über eine Bildschirmausgabe und durchläuft seinen Programmcode nur bis zu festgelegten Stoppstellen, an denen die Fortsetzung durch einen Bedienereingriff freigegeben werden muss. Für den Normalbetrieb ist die Testfunktion deaktivierbar, denn üblicherweise besitzen COM-Server der Geschäftslogik keine Benutzerinteraktionen, weil sie auf nicht nur logisch, sondern auch örtlich entfernten Rechnern ablaufen. Der Benutzer hätte keine Möglichkeit in den Ablauf einzugreifen.

Hier kann mit der Testfunktion die Arbeitsweise des Adapters illustriert werden, wie Abbildung 6-11 für das Ersetzen der Platzhalter der AWL-Quelle für FB 33 zeigt. Die linke Bildschirmmaske enthält die AWL-Quelle mit Platzhaltern (durch Anführungszeichen gekennzeichnet), in der Liste *Feldwerte* sind die Referenzen auf Felder (vgl. Abbildung 6-8) bereits durch die Originalinhalte ersetzt. Die rechte Maske stellt die fertig konfigurierte AWL-Quelle dar, bevor sie mit *Add(BausteinText, S7Source_, Quelle)* in STEP 7 eingetragen wird.

Abbildung 6-11: Substitution der Aliase im Adapter: Links übergebenes Modul; Rechts mit Ersetzungen

Die mit *Add* übergebenen Parameter bedeuten: *BausteinText*: Name des Bausteins hier: FB33; *S7Source_*: Konstante für den Typ AWL-Quelle; *Quelle*: der Inhalt der AWL-Quelle.

Die an STEP 7 übergebene Quelle FB33 kann jetzt mit STEP 7 in einen in der SPS ablauffähigen Baustein übersetzt werden. Das Kompilieren der Quelle übernimmt der SIMATIC COM-

Server *S7Source*. Im Programmcode wird deshalb die Referenz auf den Baustein an den Server S7Source übergeben. Anschließend wird der Compiler mit *S7Source.Compile* gestartet. Der Compiler legt den übersetzten Baustein FB33 automatisch in *S7Blocks* ab.

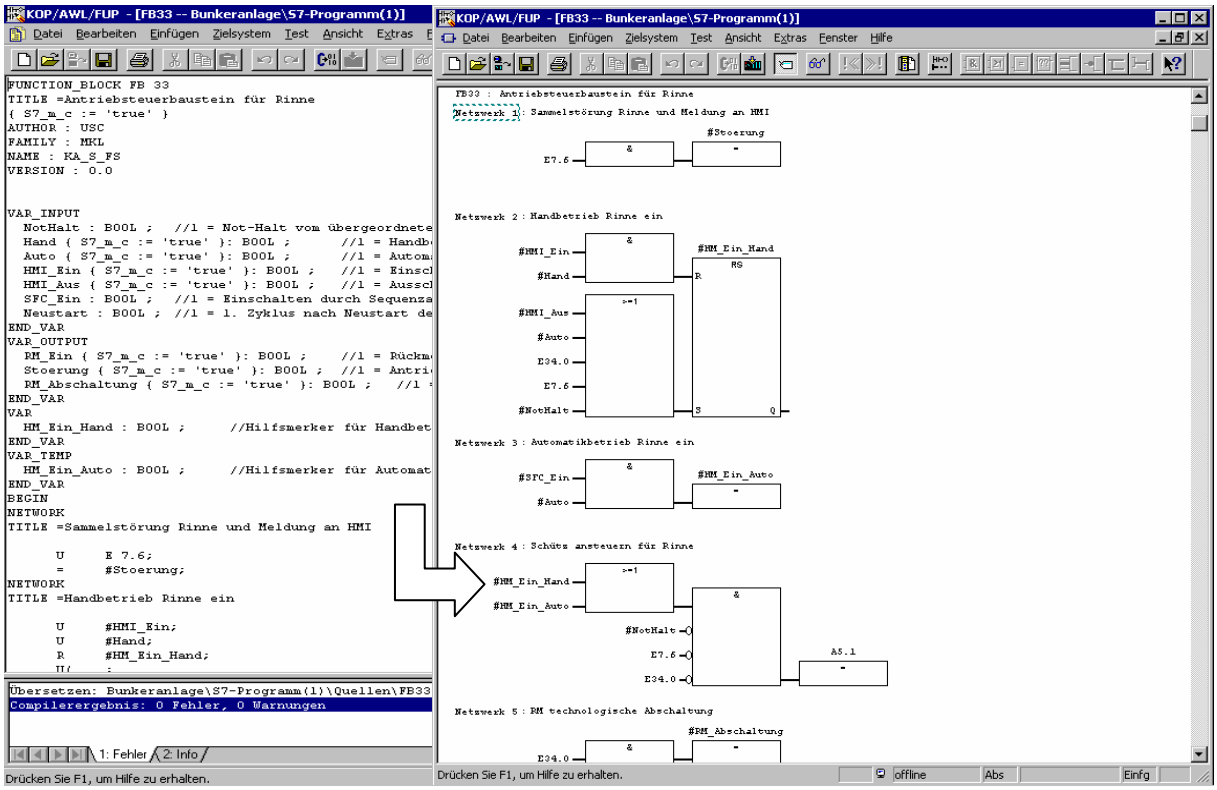

Abbildung 6-12: Links Compilerrückmeldung in STEP 7, rechts erzeugter Baustein als FUP[1]

Den Aufruf des STEP 7-Compilers für die Quelle FB33 und deren Übersetzungsmeldungen zeigt Abbildung 6-12 links. Die rechte Seite stellt den übersetzten Baustein FB33 in der Ansicht als Funktionsplan dar.

6.3 Abschlussbetrachtung

Die gewählte Keimzellenstrategie hat sich bei der Einführung als richtig erwiesen, wie die anfänglichen Schwierigkeiten bei der Definition der SPS-Module zeigten. Obwohl hier eine Anlage gewählt wurde, die nur wenige und bekannte elektro-mechanische Systeme besitzt, war es nicht einfach Module zu definieren, die unternehmensweit und für unterschiedlichste Anlagen eingesetzt werden

[1] FUP: **Fu**nktions**p**lan in STEP 7

können. Nach Überwindung der Anlaufprobleme zeigte sich, dass die Projektierung für die SPS durch Übernahme der Informationen aus der MuK-Liste vereinfacht werden konnte, weil typische doppelte Eingaben und Erfassungen entfielen.

Eine absolute Bewertung des Einsparpotenzials kann hier nicht gegeben werden, da es sich um einen Ersteinsatz handelte und der **dbEngineer** nur rudimentäre Funktionen bereitstellte. So lag beispielsweise die MuK-Liste als Excel-Tabelle vor und konnte nicht direkt importiert werden, sondern musste manuell umgeformt werden.

Gleichzeitig konnten wertvolle Anregungen für die Umsetzung des **dbEngineers** in ein Produkt gewonnen werden. Im Wesentlichen waren davon zwei Bereiche betroffen:

1. Die Handhabung und Bedienung, sowie
2. Die Funktionen.

Bei der Handhabung betrafen die meisten Wünsche eine Bedienungskompatibilität mit den Anwendungen der Officewelt, wie:

- *Verschieben von Zweigen der Aspekte*: Falls Teilanlage durch Änderungen von einem in einen anderen Zweig des Aspektebaums verschoben werden müssen, sollen sich die Aspektebäume ähnlich wie im Windows Explorer bearbeiten lassen.
- *Tabellendarstellung mit Filter*: Aktoren oder Zubehör soll in einem Tabellengitter bearbeitet und gefiltert werden können. Das vereinfacht die Änderungen ganzer Typklassen, wenn gefiltert und dann mit Excel-ähnlichen Möglichkeiten, wie Auto-Ausfüllen oder Suchen/Ersetzen-Funktionen gearbeitet werden kann (Beispiel: Alle Rinnen im Projekt erhalten einen Motor anderer Leistung).

Bei der Funktionserweiterung ging es um die Anbindung neuer Werkzeuge, sowie die Forderungen nach Import-/Exportfiltern für MuK-Listen im Officeformat.

Im Sinne der Keimzellenstrategie sollte der **dbEngineer** zuerst die Wünsche der verbesserten Handhabung erfüllen, um die Akzeptanz und Leistungsfähigkeit zu erhöhen und danach erst um zusätzliche Werkzeuge erweitert werden. Den neuen Werkzeugen und den damit in die Einführung einzubeziehenden, zusätzlichen Mitarbeitern stände die verbesserte Handhabung sofort Verfügung, was die Akzeptanz verbessert.

7 ZUSAMMENFASSUNG UND AUSBLICK

Das in dieser Arbeit vorgestellte und teilweise realisierte Konzept zur Unterstützung der Automatisierung im Bereich der Basisprojektierung zeigte, dass mit einer datentechnischen Aufbereitung und Ablage in einer Datenbank die technischen Informationen, die bei der Anlagenplanung ausgetauscht werden, nutzbringend im Sinne einer optimierten Abwicklung weiterverwendet werden können.

Die bereits von der Industrie genutzte MuK-Liste erhält durch das vorgestellte Konzept eine völlig neue Bedeutung für den Engineeringprozess als Ganzem und für die Kommunikation zwischen Anlagenbau und Automatisierungstechnik. Mit der Definition einer erweiterten MuK-Liste auf der Grundlage des offenen Dokumentenstandards XML eröffnen sich bei der Nutzung der MuK-Liste als zentraler Basis für den Datenaustausch neue Möglichkeiten. Dazu gehört die deutliche Vereinfachung beim Import in Softwaresysteme zur Weiterverarbeitung und bei der automatischen Erzeugung der Liste aus Systemen heraus. Damit findet die neudefinierte MuK-Liste Einsatzgebiete, die weit über ihre bisherige Anwendung hinausgehen.

Im Zusammenspiel der MuK-Liste mit dem durch das Konzept **dbEngineer** eingeführten modulbasierten Engineeringansatz können eine Reihe von bisher bestehenden Problemen bei der Projektierung in der Automatisierungstechnik überwunden werden. Die Qualität der mit dem **dbEngineer** generierten Unterlagen ist wegen der ausschließlichen Nutzung von geprüften und aufeinander abgestimmten Modulen höher als bei den bisherigen Vorgehensweisen. Gleichzeitig wird damit die Forderung von Unternehmen nach einer zentralen Wissensdatenbank für Projektierungslösungen erfüllt. Mit der werkzeugübergreifenden Projektierungsunterstützung entfällt die bisher notwendige getrennte Erfassung von Planungsdaten für jedes Softwarewerkzeug. Darüber hinaus wird durch die automatische Generierung von Planungsunterlagen eine Parallelität des Projektierungsprozesses erst ermöglicht, der die Gesamtdurchlaufzeit des Projekts in der Automatisierung deutlich verringert. Insgesamt erfüllt der **dbEngineer** damit wesentliche Forderungen des Marktes nach einer verbesserten Qualität der Projektierungslösung und der Verkürzung von Durchlaufzeiten in der Automatisierung bei vermindertem Aufwand in der Planung.

Durch die vollständige Unterstützung der Aspektesichten nach DIN EN 61346 durch das Konzept und der neudefinierten MuK-Liste, wird eine Verwaltung der Referenzkennzeichen für Bauteile einer Anlage und die Überführung eines Aspektes in einen anderen erst möglich, was der

Forderung der DIN EN 61346 entspricht. Mit der Realisierung von Aspektesichten gewinnt die Verständigung zwischen Anlagenbau und Automatisierung eine neue Qualität, die mit den bekannten Vorgehensweisen nicht erreichbar ist.

Mit der Entscheidung als Softwarearchitektur des **dbEngineers** das COM/COM+-Modell einzusetzen, wurde die nötige Flexibilität und Skalierbarkeit für den Einsatz vom Einzelplatzsystem bis zur Installation als verteilte Applikation mit einer Vielzahl von Clients und mehreren Servern geschaffen. Mit der durch COM gleichzeitig nutzbaren ADO-Technologie hat der Anwender größtmögliche Freiheitsgrade bei der Wahl seines DBMS. Er kann entsprechend seiner Leistungsanforderungen von einer Desktop-Datenbank bis hin zu DBMS, die auf verteilten Servern betrieben werden, alle marktgängigen Systeme einsetzen.

Gleichzeitig kann durch die Spezifikation der COM-Schnittstelle für Adapter das System **dbEngineer** von Anwendern, Lieferanten oder Softwarehäusern, um Adapter mit unterschiedlicher Leistungsfähigkeit erweitert werden. Das Konzept bietet durch seine Erweiterbarkeit das Potenzial, den Nutzen des **dbEngineers** deutlich über die bisher gezeigten Einsatzfälle hinaus weiter zu entwickeln und dem Planungsprozess eine völlig neue Qualität zu geben. Denkbar ist beispielsweise, dass Funktionen, die ein Planungswerkzeug über seine Schnittstelle nicht anbietet, vom Adapter intern bereitgestellt werden, wodurch Softwarehäuser sich im Wettbewerb ein Alleinstellungsmerkmal schaffen können. Andererseits besteht die Möglichkeit neue oder weitere, in der Arbeit bisher nicht genannte, häufig genutzte Werkzeuge oder Hilfsprogramme, wie Kabel- oder Kurzschlussstrom-Berechnungsprogramme über eigene Adapter einzubinden oder den **dbEngineer** bereits in der Angebotsphase eines Projekts für die Ermittlung von Mengengerüsten einzusetzen.

Der vom **dbEngineer** eingeführte und unterstützte modulbasierte Projektierungsansatz wird in zukünftigen Automatisierungslösungen einen nicht mehr wegzudenkenden Stellenwert besitzen. Neuere Entwicklungen, wie PROFInet[1] [39] und die darauf aufgebaute Component based Automation[2] (CbA) [40], ein Projektierungsansatz für die SPS-Programmierung, die sich mit der Interaktion und Verschaltung von Grundsystemen beschäftigen, die oberhalb von **dbEngineer** ablaufen, set-

[1] PROFInet: Systemspezifikation für dezentrale und verteilte Automatisierungslösungen auf Basis von Feldbussystemen, veröffentlicht von der PROFIBUS Nutzerorganisation e.V.; Karlsruhe

[2] Component based Automation (engl.): Komponenten basierte Automatisierung

zen die Existenz von geeigneten Modulen für ihre Zwecke voraus. Der **dbEngineer** ist dabei hervorragend geeignet, die Basisplanung abzudecken und verschaltbare SPS-Module zu generieren, die dann mit Hilfe von PROFInet und CbA zu Gesamtsystemen der Automatisierung weiterverarbeitet werden.

Das in dieser Arbeit vorgestellte Konzept stellt einen Beitrag zur gegenwärtigen Diskussion in der Automatisierungstechnik über die Integration der unterschiedlichen Werkzeuge und der automatisierten Erstellung von Planungsunterlagen dar. Wie aber gerade das vorgestellte Konzept des **dbEngineers** zeigt, kann es bei allen diesen Entwicklungen nur darum gehen, den Menschen von standardisierten Routinetätigkeiten zu entlasten, um ihm den Freiraum zu schaffen, den er benötigt, um seine Kreativität in die Entwicklung völlig neuer Lösungen einbringen zu können.

LITERATURVERZEICHNIS

1 Esser, W.: Softwarelösungen erhöhen die Wirtschaftlichkeit automatisierter Maschinen und Anlagen, atp Oldenbourg Verlag Heft 9, 2002

2 Schütten, M.: Konzept eines COM-basierten Technischen Informationssystems (TPIS), Universität Essen, Lehrstuhl für Ingenieurinformatik, Dissertation 2001

3 Ahrens, W.; Klein, W.; Rauprich, G. und Schellberg, M.: Enterprise Application Integration (EAI), atp Oldenbourg Verlag Heft 12, 2002

4 Deutscher Maschinenbau braucht in Zukunft qualifizierte Ingenieure: Kurzfassung einer Untersuchung der Prognos AG im Auftrag der IMPULS-Stiftung des VDMA, www.vdma.de, 22. März 2002

5 VDI/VDE Richtlinie 3694, Lastenheft/Pflichtenheft für den Einsatz von Automatisierungssystemen, April 1991

6 Anleitung zur Motoren- und Komponentenliste: Begriffe – Vordrucke –Erläuterungen Thyssen Stahl AG

7 TH-N 204 570, Werknorm der ThyssenKrupp Stahl AG, Duisburg

8 Ahrens, W.; Buchner, H.; Rauprich, G. und Scheurlen, H.-J.: INTERKAMA 1999: CAE-Systeme für die Prozessleittechnik Teil 1: Systeme, atp Oldenbourg Verlag Heft 3, 2000

9 DIN EN 61346-1 Januar 1997, Industrielle Systeme, Anlagen und Ausrüstungen und Industrieprodukte: Strukturierungsprinzipien und Referenzkennzeichnungen, Teil 1: Allgemeine Regeln

10 Lauber R., Göhner P.: Prozessautomatisierung 1, 3. Auflage 1999, Springer Verlag Berlin Heidelberg New York

11 Früh, K. F. (Hrsg.): Handbuch der Prozessautomatisierung, Oldenbourg Verlag 1997

12 Hahn, R.: SIMATIC Erfolg mit System, Publicis MCD Verlag, 2001

13 TH-N 400 020-3, Werknorm der ThyssenKrupp Stahl AG, Duisburg

14 Wie liest man deutsche Stromlaufpläne für Industrieausrüstungen?, E59C Information, Siemens AG Berlin und München, Bereich Energie- und Automatisierungstechnik, Ausgabe 1985

15 DIN EN 61131-3, Ausgabe:1994-08 Speicherprogrammierbare Steuerungen - Teil 3: Programmiersprachen (IEC 61131-3:1993); Deutsche Fassung EN 61131-3:1993

16 PLCopen, Certified Products, http://www.plcopen.org/certification/certified_products.htm, April 2003

17 Berger, H.: Automatisieren mit STEP 7 in KOP und FUP, Publicis MCD Verlag, 1999

18 Angerbauer, R.: Baukastensystematik in der Steuerungstechnik auf der Basis einer Förderalen Informationsarchitektur, 2002, veröffentlicht unter: www.foederal.org.

19 Programmer´s Reference Delphi 6, Borland GmbH Langen

20 Kosch, A.: Crashkurs .NET für Delphianer, Software & Support Verlag GmbH, 2003

21 Kosch, A.: COM/DCOM/COM+ mit Delphi, Software & Support Verlag GmbH, 2000

22 DIN EN 28879, Ausgabe:1991-03: Informationsverarbeitung; Textverarbeitung und -kommunikation; Genormte Verallgemeinerte Auszeichnungssprache (SGML) (ISO 8879:1986 + A1:1988); EN 28879:1990

23 W3C: Extensible Markup Language (XML), veröffentlicht unter: http://www.w3.org/XML/#intro.htm

24 Ray, E. T.: Einführung in XML, 1. Auflage 2001, O`Reilly Verlag Köln

25 Sauer, H.: Relationale Datenbanken, 5. Auflage, Addison-Wesley Verlag, 2002

26 STEP7 V5.1 Produkt-CD, Online Handbuch „Kommandoschnittstelle“, Siemens AG, Berlin und München, Siemens-Nr.: S79220-A6320-F000-01, 2002

27 Siemens AG: Schalten, Schützen, Verteilen in Niederspannungsnetzen; Handbuch mit Auswahlkriterien und Projektierungshinweisen für Schaltgeräte, Steuerungen und Schaltanlagen, 4. Auflage 1997, Publicis MCD Verlag

28 ISO/IEC 9579, Ausgabe:2000-02 Informationstechnik - Fernzugriff auf Datenbanken für SQL mit Sicherheits-Erweiterungen

29 ISO/IEC 9075-1, Ausgabe:1999-12 Informationstechnik - Datenbanksprachen - SQL - Teil 1: Rahmenwerk (SQL/Rahmenwerk),

30 Eplan Software & Service: Schaltplangenerator automatisiert Engineering, IEE 47, Hühtig Verlag, Heft 3, 2002, S24ff

31 Menzel, G.; Freier, D.; Adam, G.; Hauff, T.: Investitionssicherheit von Einrichtungen der Prozessleittechnik eine (un)lösbare Aufgabe?, atp Oldenbourg Verlag Heft 3, 2003

32 Platform SDK: Windows Installer, MSDN Library, Feb 2003, veröffentlicht unter: http.//www.microsoft.com

33 Rauprich, G.; Haus, C.; Ahrens, W.: PLT-CAE – Integration in gewerkeübergreifendes Engineering und Plant-Maintenance, atp Oldenbourg Verlag Heft 4, 2002

34 Platt, D. S.: Microsoft .NET – Eine Einführung, 2. erweiterte Auflage, Microsoft Press 2002

35 Beiblatt 1 zu DIN EN 61346-1 November 2002, Industrielle Systeme, Anlagen und Ausrüstungen und Industrieprodukte: Strukturierungsprinzipien und Referenzkennzeichnungen, Anwendungsleitlinien

36 Beiblatt 2 zu DIN EN 61346-1 Februar 1999, Industrielle Systeme, Anlagen und Ausrüstungen und Industrieprodukte: Strukturierungsprinzipien und Referenzkennzeichnungen, Betrachtung von Begriffen und deren Zusammenhänge

37 DIN EN ISO 9001, Ausgabe: 2000-12, Qualitätsmanagementsysteme – Anforderungen (ISO 9001:2000-09); Dreisprachige Fassung EN ISO 9001:2000

38 Schäfer, D.; Roller, D.: Electrical Engineering Solutions – ECAE Systeme der dritten Generation, atp Oldenbourg Verlag Heft 3, 2003

39 PROFInet, PROFIBUS Nutzerorganisationen e.V., http://www.profibus.com/libraries.html, Mai 2003

40 Component based Automation (CbA), Siemens AG Berlin und München, http://www.ad.siemens.de/cba/index_00.htm, Mai 2003