

STUDIENARBEIT

Joachim Mack

Studiengang: Informatik
Prüfer/in: Prof. Dr. Dieter Roller
Betreuer/in: Dipl.-Math. Dirk Schäfer
Begonnen am: 1.10.2001
Beendet am: 31.3.2002
CR-Nummer: J2, J6

Studien/Diplomarbeit-Nr. 1827

REALISIERUNGSKONZEPT FÜR DIE
IMPLEMENTIERUNG EINES
SOFTWARE-PROTOTYPEN
FÜR EIN ECAD VARIANTENMODUL

Joachim Mack

Inhaltsverzeichnis:

1	EINLEITUNG UND PROBLEMSTELLUNG.....	4
1.1	STAND DER TECHNIK.....	4
2	VARIANTENTECHNOLOGIE.....	6
2.1	VORTEILE DER VARIANTENTECHNOLOGIE.....	7
2.2	KOMPONENTENKONSTRUKTION.....	8
2.3	MODULANSATZ.....	10
2.4	BESCHREIBUNG DES ZU ÜBERARBEITENDEN KONZEPTS.....	11
3	VARIANTENKONSTRUKTIONSMODUL.....	13
3.1	DIE DATENSTRUKTUR.....	14
3.2	KONFIGURATOR.....	15
3.3	KOPPLUNGSMODUL.....	17
3.4	INTERFACE.....	18
3.5	VORGEHENSWEISEN BEI DER REALISIERUNG.....	19
4	GRUNDLAGEN DER INFORMATIK.....	20
4.1	GRAMMATIKEN.....	20
4.2	GRUNDLAGEN DES COMPILERBAUS.....	26
4.2.1	<i>Analyse</i>	28
4.2.2	<i>Synthese</i>	30
4.3	SOFTWARE ENGINEERING.....	31
4.3.1	<i>Ziele des Software Engineerings</i>	31
4.3.2	<i>Der Software-Life-Cycle</i>	32
4.3.3	<i>Aufwandsverteilung auf die Phasen der Software-Entwicklung</i>	34
4.3.4	<i>Zusammenhang zwischen Fehlerentstehung und -entdeckung</i>	34
4.3.5	<i>Analyse</i>	35
4.3.6	<i>Spezifikation</i>	35
4.3.7	<i>Entwurf</i>	36
4.3.8	<i>Codierung</i>	36
4.3.9	<i>Test</i>	37
4.4	STANDARDS UND NORMEN.....	38
4.4.1	<i>Einige Datenaustausch-Normen im Bereich Elektrotechnik CAD</i>	38
4.4.2	<i>IGES</i>	39
4.4.3	<i>SET</i>	41
4.4.4	<i>VNS</i>	42
4.4.5	<i>EDIF</i>	42
4.4.6	<i>Internationaler Standard für Produktdatenaustausch STEP</i>	43
4.4.7	<i>Anwendungsprotokoll 212 (AP 212)</i>	45
5	AUSWAHL DER ENTWICKLUNGSUMGEBUNG.....	48
5.1	SYSTEME.....	48
5.1.1	<i>Betriebssysteme</i>	48
5.1.2	<i>ECAD-Systeme</i>	51
5.2	PROGRAMMIERSPRACHEN UND WERKZEUGE.....	52
5.2.1	<i>C++</i>	52

5.2.2	<i>Java</i>	53
5.2.3	<i>VCL</i>	56
5.2.4	<i>XML</i>	57
6	ANWENDUNGS-DEMOBEISPIEL	67
6.1	STATISCHE KONFIGURATION.....	71
6.2	DYNAMISCHE KONFIGURATION.....	71
6.3	REALISIERUNGSASPEKTE.....	72
6.4	MÖGLICHE PROBLEME UND RISIKEN	72
7	SCHLUSSFOLGERUNG UND AUSBLICK	74
8	LITERATUR	76
9	INDEX	78

1 Einleitung und Problemstellung

Die in der Elektroindustrie verwendeten CAD-Systeme¹, genannt ECAD-Systeme, entwickeln sich permanent weiter. Die Faktoren Zeitreduzierung und Kostensenkung treiben ECAD-Systemhersteller immer wieder dazu an, neuere und effizientere Techniken in ihren Produkten zur Verfügung zu stellen. Die heute auf dem Markt verfügbaren ECAD-Systeme bieten zwar eine Vielzahl an Möglichkeiten, die den Konstrukteur bei seiner Arbeit unterstützen sollen, jedoch gibt es einige Bereiche, die bisher vernachlässigt wurden [1]. Einer dieser Bereiche ist die Unterstützung der Anwender durch Variantentechnologie². Hier wurde im Gegensatz zum Mechanischen-, bzw. Maschinenbau- CAD Bereich (MCAD) bisher keine befriedigende Lösung angeboten.

Ein möglicher Ansatz für die Entwicklung eines Variantenkonstruktionsmoduls für den Bereich ECAD wurde im Rahmen einer Kooperation zwischen der Universität Stuttgart und einem ECAD-Systemhersteller erarbeitet [2]. Der für die Umsetzung dieses Konzeptes veranschlagte Realisierungsaufwand war allerdings so hoch, dass bisher mangels Ressourcen auf eine praktische Umsetzung des Konzeptes verzichtet werden musste.

Die Studienarbeit soll das bestehende Konzept überarbeiten, und Möglichkeiten für eine prototypische Realisierung des Konzepts im Rahmen einer Diplomarbeit aufzeigen. Ferner soll sie als Grundlage für eine spätere, eventuell kommerzielle, vollständige Realisierung dienen.

Einleitend wird dem Leser eine kurze Einführung in Variantenkonstruktion im ECAD und in das bestehende Konzept gegeben. Danach wird das überarbeitete Konzept vorgestellt. Im Anschluss daran bekommt der Leser die zur Realisierung notwendigen Grundlagen der Informatik vermittelt. In den folgenden Kapiteln wird auf die für eine Entwicklung generell in Frage kommende Entwicklungsumgebungen, Werkzeuge, Programmiersprachen und Betriebssysteme eingegangen. Basierend auf diesem Fundament wird die Studienarbeit durch eine elementare Beispielkonstruktion, anhand derer die Funktionsweise des prototypischen Variantenmoduls später demonstriert werden soll, abgerundet.

1.1 Stand der Technik

Bevor im Kapitel 1 auf Variantentechnologie näher eingegangen wird, muss zunächst der Stand der Technik der ECAD-Systeme und ihrer Entwicklungsstufen analysiert werden. Die dem heutigen „State of the Art“ entsprechenden ECAD-Systeme gehören zur so genannten dritten Systemgeneration.

Die Systeme der ersten und zweiten Generation, die auch heutzutage noch stark verbreitet sind und eingesetzt werden, weisen schwerwiegende Probleme auf. Diese sind unter anderem:

- Häufig wird mit einem ECAD System lediglich gezeichnet. Trotz der vielen Funktionalitäten, die ein ECAD-System dem Konstrukteur zur Verfügung stellt, beschränken sich viele Konstrukteure darauf, lediglich Stromlaufpläne damit zu erzeugen. Vor allem in der Anfangszeit des ECAD spielte die Tatsache eine

¹ CAD-System: Computer Aided Design, rechnergestützter Entwurf

² Variantentechnologie: siehe Kapitel 1

Rolle, dass ein erfahrener Konstrukteur anfangs eine Zeichnung konventionell schneller erstellen konnte, als mit Hilfe eines CAD-Systems. Manch ein Konstrukteur erkannte die Vorteile und den Nutzen der CAD-Systeme nicht sofort, da er anfänglich den Umgang mit dem CAD-System, als „lästig“ und „arbeitszeitverlängernd“ empfand. Nicht zu vergessen ist die Tatsache, dass in der Vergangenheit oft die notwendige Einarbeitungszeit und das erforderliche Training der Konstrukteure, welches unter allen Umständen erforderlich ist, aus Kostengründen vernachlässigt wurden [3].

- Jede Modifikation an einem Projekt muss in einem konventionellen ECAD System meist einzeln und blattweise durchgeführt werden, was sich als eine sehr mühsame Tätigkeit erweisen kann.
- Situationen, bei denen aufgrund einer einzigen, eigentlich geringfügigen Konstruktionsänderung der gesamte Stromlaufplan neu gezeichnet werden muss, sind für die heutigen ECAD-Systeme nicht effizient handhabbar und müssen in der Regel manuell, wie oben beschreiben, durchgeführt werden. Dies führt zu einer erhöhten Fehleranfälligkeit nach Konstruktionsänderungen, da die durchgeführten Modifikationen schwer zu überblicken und damit auch schwer zu kontrollieren sind.
- Die Wiederverwendung bestehender Projekte bzw. Konstruktionen beschränkt sich überwiegend auf Kopieren und manuelles Abändern.
- Die Varianten, die durch manuelles Kopieren erzeugt werden, müssen vollständig abgespeichert werden, was einen erhöhten Speicherbedarf zur Folge hat.

Diese Probleme sollen in Systemen der dritten Generation abgestellt werden. Ein schlichtes Erweitern der Systeme der zweiten Generation um Zusatzfunktionalitäten kann diese Probleme nicht befriedigend lösen, weil man bei der Entwicklung kaum „Software Engineering“ Methoden eingesetzt hat. Deshalb ist eine komplette Neuentwicklung bei Systemen der dritten Generation unumgänglich.

Zusätzlich müssen ECAD Systemhersteller durch den steigenden Wettbewerbsdruck und die zunehmende Globalisierung immer mehr innovative Ideen in ihre Produkte integrieren. Langfristig gesehen sind Unternehmen, die neue Technologien vor anderen Unternehmen einführen, deutlich im Vorteil. Deshalb wird die Variantentechnologie im Bereich der Elektrokonstruktion und Anlagenprojektierung als eine der vielversprechendsten angesehen.

2 Variantentechnologie

Die Konstruktionen von Produktvarianten eines Herstellers im ECAD-Bereich unterscheiden sich oft kaum voneinander, müssen allerdings nahezu komplett von Hand neu konstruiert werden. Um in diesem Bereich eine größere Effizienz zu erreichen, müssen zukünftige ECAD-Systeme hier eine entsprechende Hilfestellung bieten. Die meisten derzeit verfügbaren ECAD-Systeme bieten bisher nur eine marginale Unterstützung für die Lösung dieses Problems an. Sie beschränken sich überwiegend auf die „Wiederverwendung“ von bestehenden Konstruktionen, indem sie das Kopieren von Teilprojekten und anschließendes Einfügen derselben in eine neue Konstruktion unterstützen. Die neue Konstruktion kann dann als neues Projekt gespeichert werden. Ein leistungsfähiges ECAD-System sollte heutzutage aber in der Lage sein, auf Wunsch des Konstrukteurs aus allen gespeicherten Projekten einzelne Konstruktionsbestandteile unterschiedlicher Granularität in ein neues Projekt zu importieren, und ihm so zu ermöglichen, schnell und effizient Varianten eines Produkts bzw. neue Produkte mit Hilfe von schon vorhandenen Teilkonstruktionen zu erstellen.

Seit geraumer Zeit wird in MCAD-Systemen Variantentechnik effizient eingesetzt [4]. Die aus der Verwendung von Variantentechnologie resultierenden Vorteile, wie Kosten- und Zeitreduzierung, sollen künftig auch für die Bereiche Elektrokonstruktion und Anlagenprojektion nutzbar gemacht werden. Die Variantenkonstruktion im Bereich der Elektrotechnik unterscheidet sich gravierend von der im Bereich des Maschinenbaus. MCAD orientiert sich im Wesentlichen an der Geometrie der Konstruktionen [5]. Im ECAD hingegen sind es logische Informationen, die primär eine Rolle spielen. Insofern ist es für die Variantenkonstruktion im ECAD wichtiger Änderungen an Konstruktionen automatisch durchzuführen, als Geometrieänderungen zu verwalten. Aus diesem Grund kann man die Variantentechnologie aus MCAD nicht einfach ins ECAD übernehmen, sondern muss sie von Grund auf neu entwickeln.

Für den Einsatz von Variantentechnologie im Bereich der Elektrokonstruktion bieten sich eine effiziente Konstruktion von elektrotechnischen Produktfamilien, sowie eine konsistente Datenhaltung von Konstruktionen an. Diese kann, sofern ECAD Systemhersteller diese Funktionalität nicht in den Kernsystemen anbieten, beispielsweise mit einem Variantenmodul gelöst werden, das mit einem ECAD System kommuniziert. Ferner wird ein neues Datenmodell für den Bereich ECAD benötigt, das die angestrebte Wiederverwendbarkeit ermöglicht und die effiziente Modifikation von bestehenden elektrotechnischen Konstruktionen erleichtert. Das Ziel ist es, die Konstruktionszeiten zu reduzieren, um damit die Konstruktionskosten erheblich zu senken.

Da sich der Anteil der Neukonstruktionen am Gesamtkonstruktionsaufwand lediglich auf ca. 10-15% beläuft, der Rest sich jedoch aus Änderungs- und Variantenkonstruktionen zusammensetzt [6], wird die Einführung der neuen Technologie einen deutlichen Fortschritt bringen.

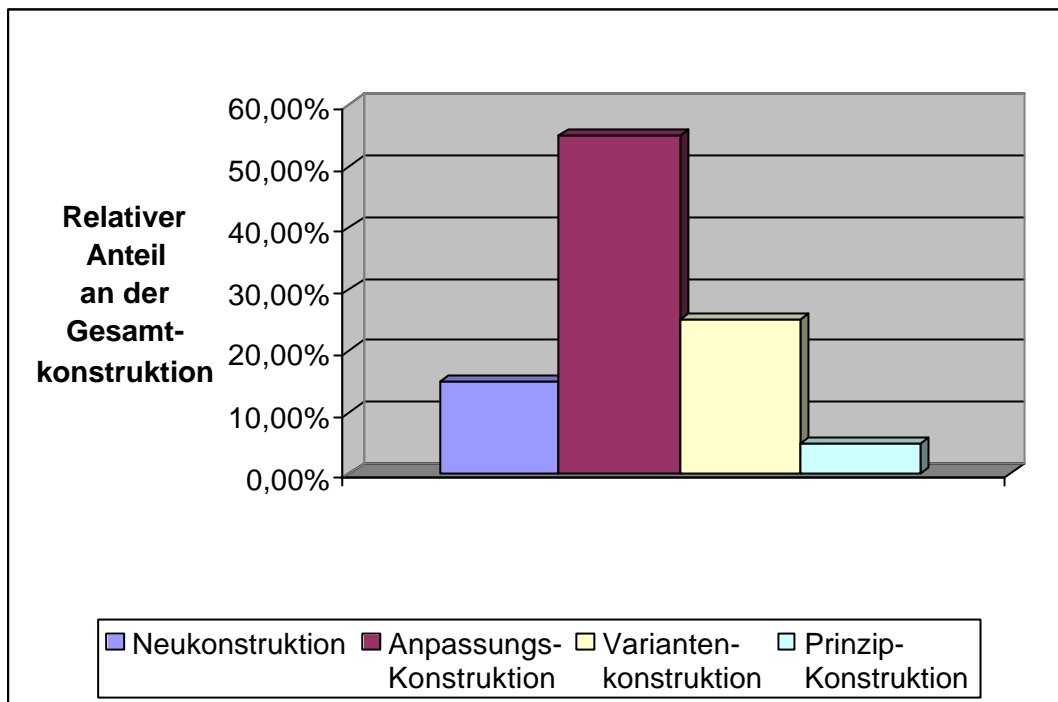


Bild (1) Relative Verteilung der Konstruktionsarten (in Anlehnung an [5])

Der Einsatz der Variantentechnik im MCAD-Bereich hat ein enormes Kostensenkungspotential aufgezeigt, das auch im ECAD-Bereich prinzipiell vorliegt. Gleichzeitig ist eine erhebliche Verbesserung der Konstruktionsqualität zu erwarten. Die Variantentechnologie kann als Herzstück einer intelligenten Komponententechnik angesehen werden, über die eine auf standardisierten Funktionsmodulen basierende modulbaukastenähnliche Konstruktionsweise ermöglicht wird.

Die Variantentechnik findet auch im Bereich des Produktmarketing Anwendung. Man kann bereits hier dem Kunden einen ungefähren Preisvorschlag unterbreiten, da die standardisierten Großmodule sich schon in der Datenbank befinden. Dem Kunden können so verschiedene Konfigurationen angeboten und die daraus resultierenden Kosten genannt werden. Er kann direkt vor Ort auswählen und bestellen.

2.1 Vorteile der Variantentechnologie

Mittels der Variantentechnologie können die Probleme der Wiederverwendbarkeit, der Modifizierbarkeit und der konsistenten Datenhaltung von Konstruktionen zum Vorteil gegenüber konventionellen ECAD-Systemen, effektiv gelöst werden.

Die Vorteile, die sich aus dem Einsatz der Variantentechnologie ergeben, sind unter anderem [4], [7]:

- Wiederverwendbarkeit
Im Gegensatz zu konventionellen ECAD-Systemen ist die Wiederverwendbarkeit im Variantenmodul nicht mehr als einfaches Kopieren anzusehen. Es werden hierbei mit dem Kopieren von Komponenten des Stromlaufplanes auch alle dazugehörigen logischen Daten und Informationen kopiert. Eine neue Konstruktion kann einfacher und schneller erzeugt werden, indem man in einer bestehenden Konstruktion lediglich die

Parameterwerte abändert. Damit kann man zum Beispiel aus bestehenden Standardstromlaufplänen funktionsfähige Einzelstromlaufpläne erzeugen.

- **Effiziente Konstruktion von Produktfamilien**
Durch geringfügige Änderungen aus bestehenden Konstruktionen kann man Varianten der gewählten Produktfamilie erzeugen.
- **Kompakte Speicherung**
Die Speicherung aller Varianten als separate Dateien einer Produktfamilie ist nicht mehr erforderlich. Es genügt, eine Musterkonstruktion abzulegen, und für die jeweiligen Varianten die Parameter zu speichern. Dies führt zu einer erheblichen Einsparung an Speicherplatz.
- **Grobentwurf von Konstruktionen**
Bei manchen Konstruktionen stehen relevante Informationen oft nicht fest. Der Einsatz von Parametrik erlaubt es dem Konstrukteur, die Daten später in das System einfließen zu lassen.
- **Effiziente Modifikationsmöglichkeiten**
Wenn „nur“ Parameter geändert werden müssen, sind die Änderungsanpassungen leichter handhabbar. Die Restriktionen zwischen den elektrotechnischen Betriebsmitteln sind im Variantenmodul definiert.
- **Schnelle Konstruktion durch Komponententechnik.**
- **Verbesserte Fehlerkontrolle in Stromlaufplänen.**

2.2 Komponentenkonstruktion

Eine spezielle Art Variantentechnologie einzusetzen, ist die Komponentenkonstruktion. Das Grundprinzip der Komponentenkonstruktion ist die Wiederverwendung bereits definierter Module mit festgeschriebener Kombinationsfähigkeit. Hierzu ist eine Verfeinerung der Definition von Komponenten im ECAD-Bereich notwendig.

Für die prototypische Implementierung im Rahmen des Forschungsprojekts eignet sich aus Zeitgründen folgendes Vorgehen: Im CAD-System wird der jeweilige Bereich markiert, der später als Komponente dargestellt werden soll. Der Bereich außerhalb der Markierung wird gelöscht, oder aber der markierte Bereich wird kopiert, und in ein leeres Blatt eingefügt. Somit existiert auf diesem Blatt lediglich eine Komponente. Diese wird nun in ein Zwischenmodul des Variantenmoduls exportiert, woraufhin dieses die neue Komponente im Frontend für den Benutzer sichtbar macht. Beim Export der Daten aus dem ECAD-System sollten Grunddaten wie Autor, Erstelldatum und interne ID automatisch zugewiesen bzw. ausgelesen werden. Des Weiteren müssen in einem Zwischenschritt die Daten in eine geeignete Form transformiert und in einer Datenbank gesichert werden. Hier wäre denkbar, den Anwender die

Daten durch eine Interaktion über eine GUI-Anwendung mit Parametern versehen zu lassen. Der Benutzer sollte dann die restlichen Parameter einstellen.

Folgende Arbeiten sollten durchgeführt werden, um die funktionale Abgeschlossenheit, Funktionsfähigkeit und Korrektheit sicherzustellen:

- Hierarchien in den Komponenten anpassen
- Zusätzliche Betriebsmittel aufnehmen oder überzählige entfernen
- Querverweise überprüfen, gegebenenfalls korrigieren
- Funktionsfähigkeit der Komponente sicherstellen
- Ausführliche Beschreibung der Komponente sicherstellen
- Klassifizierung der Komponente durchführen
- Komponente in einer Bibliothek ablegen
- Die Komponente prüfen und freigeben (elektronische Unterschrift)

Ferner ist es von Vorteil, wenn folgende, weitere Attribute definiert werden:

- Wartungskosten (in €/Monat)
- Zuverlässigkeit (hoch, mittel, mäßig, niedrig)
- Handling (gut,..., schlecht)
- Vorkalkulation (Gesamtkosten bis Inbetriebnahme: €613.232)
- Aufbau (einfach, kompliziert, kritisch,...)
- Ersatzteile (in €/Jahr)

Dadurch kann eine Komponente, die nun aus dem Projekt herausgelöst ist, in anderen Projekten universell eingesetzt werden. Die Komponente sollte jedoch den Namen des Ursprungsprojekts festhalten, um Änderungen an der Originalkonstruktion nicht auf die herausgelöste Komponente überfließen zu lassen. Diese wird an eine Komponentenverwaltung weitergereicht.

Möglichkeiten der Komponentenverwaltung:

- Komponenten, die bereits existieren, abrufen
- Parameter belegen
- Kategorien anlegen
- In- und Output-Schnittstellen (im Stromlaufplan) definieren
- Randbedingungen (Toleranzen) festlegen

Ferner sollte es die Möglichkeit geben, Kompatibilitätsmatrizen für Komponenten und Klassen zu generieren, in denen festgehalten wird, für welche Zwecke sie verwendet werden können, oder nach dem Autor zu sortieren. Komponenten, die so in der Komponentenbibliothek abgelegt werden, können als fehlerfrei und funktional abgeschlossen angesehen werden.

Eine Verbesserung der Effizienz kann dadurch erreicht werden, dass jede Komponente mindestens einer Kategorie bzw. Klasse angehört. Diese Kategorien können nun im Komponentenbrowser angezeigt werden und damit dem Benutzer die Auswahl der Komponenten erleichtern.

2.3 Modulansatz

Ein modular aufgebautes ECAD – System sollte aus einem Grundpaket und mehreren darauf aufbauenden Zusatzmodulen bestehen. Dies bietet dem Anwender die Möglichkeit, ein individuelles, auf seine Anforderungen zugeschnittenes Paket zusammenzustellen. Nicht benötigte Funktionen müssen so nicht mehr bezahlt werden, und man kann trotzdem sein System entsprechend der eigenen Anforderungen erweitern oder abändern. Weitere Vorteile, die sich durch den modularen Aufbau ergeben, sind hohe Wiederverwendbarkeit und geringer Wartungsaufwand.

Da der Anwender oft eigene Probleme durch Einbindung selbstgeschriebener Programme in das System löst, sollte es ihm möglich sein, diese Lösungen auch anderen Kunden als vollwertige, vom Systemhersteller anerkannte Module, zur Verfügung zu stellen. Um die Stellung der Software auf dem Markt zu festigen, ist daher der Modulentwurf durch einen OEM Partner eine effiziente Lösung.

Eine firmenweite Integration des ECAD – Systems kann nur dann erfolgen, wenn die Schnittstellen zu allen anderen Organisationssystemen definiert sind. Dies ist unabdingbar, da im Rahmen der Globalisierung dieser Aspekt immer wichtiger wird.

ECAD-Systeme sollten die Möglichkeit bieten, externe, unabhängige Werkzeuge ins das System einbinden zu können. Dadurch wird zum einen dem Anwender eine preisgünstige Erweiterungsmöglichkeit geboten, zum anderen könnten ECAD-Systemhersteller davon profitieren, indem sie die Technologien dieser Tools ins ECAD-System adaptieren.

2.4 Beschreibung des zu überarbeitenden Konzepts

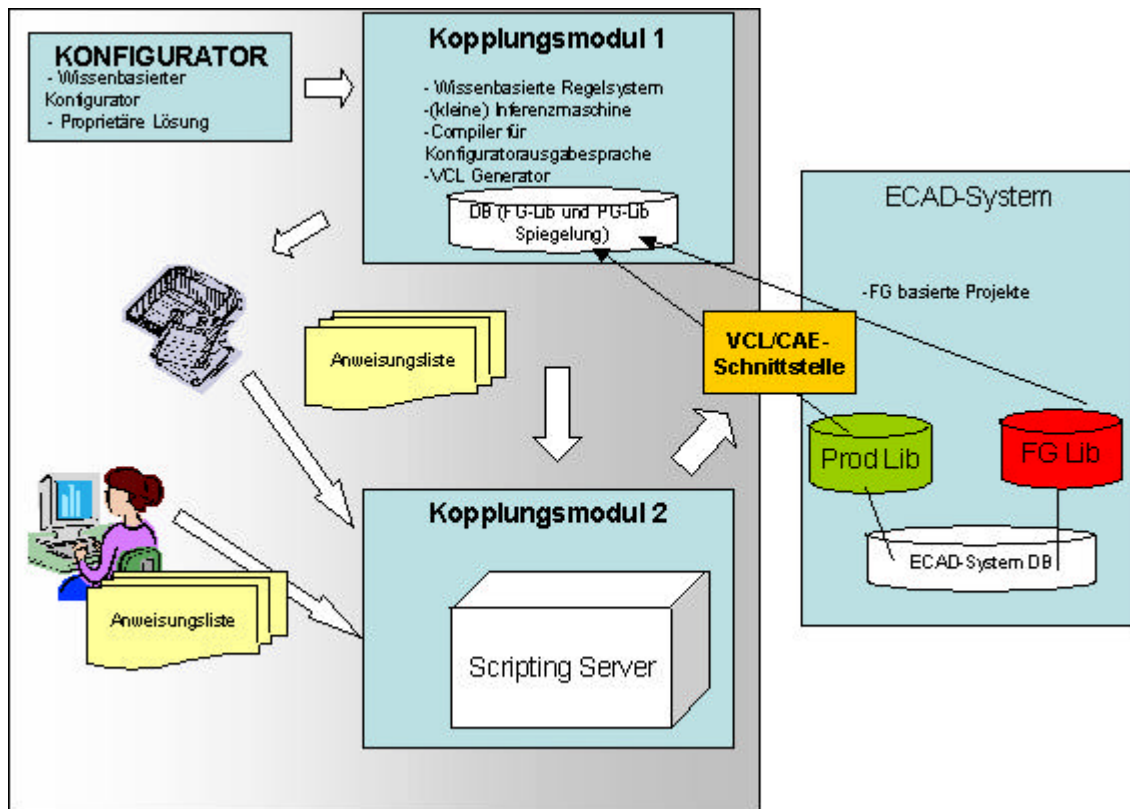


Bild (2) zu überarbeitendes Konzept

Wie in der Einleitung bereits erwähnt, handelt es sich bei dem zu überarbeitenden Konzept um die Entwicklung eines Variantenkonstruktionsmoduls für den Bereich ECAD im Rahmen einer Kooperation zwischen der Universität Stuttgart und einem ECAD-Systemhersteller.

Die Möglichkeit, mit Komponenten zu arbeiten, bildet das Herz des Variantenmoduls. Komponenten können in Projekten wieder verwendet werden. Sie stellen nach außen ihre Schnittstellen und Funktionalitäten zur Verfügung, sind allerdings im Aufbau ihrer Datenstruktur dem Anwender völlig unbekannt. Man spricht hier vom klassischen Blackbox Prinzip.

Dieses Konzept für ein Variantenmodul besteht aus einem Konfigurator, einem Kopplungsmodul 1 und Kopplungsmodul 2. Der Konfigurator ist eine GUI, in der sich wissensbasiert Produktvarianten zusammenstellen lassen. Diese werden durch ein wissensbasiertes Regelsystem überprüft und anschließend in eine Metasprache transformiert. Die Funktion des zweiten Kopplungsmoduls besteht im Wesentlichen darin, Befehle in eine für das ECAD-System verständliche Form aufzubereiten. Das Variantenmodul verfügt außerdem über eine Schnittstelle, mittels der mit ihrem ECAD-System kommuniziert wird.

An dieser Stelle sollte man hinzufügen, dass nicht unbedingt eine komplette Umstellung der vorhandenen Daten auf die parametrischen Modelle notwendig ist. Die nicht parametrisierten Daten können ebenso noch weiter benutzt werden. Die Komponenten können durch das Variantenmodul nach Bedarf definiert werden, oder man kann bereits definierte Komponenten in jedem Projekt wiederverwenden.

Die Nachteile des alten Konzepts liegen darin, dass es maßgeschneidert wurde um in das bestehende Produkt eines ECAD-Systemherstellers nahtlos integriert zu werden. D.h. es wäre nur lauffähig in Verbindung mit dessen Software. In der ursprünglichen Konzeption ist man nicht unabhängig vom ECAD-System, mit dem zusammengearbeitet wird. Man müsste dort Zugriff auf den Programmcode des ECAD-Kernels haben, was sich in der Realität als nicht durchführbar erweist, weil jeder Systemhersteller darauf bedacht ist, anderen keinen Einblick in seine Technologien zu geben. Das ist aber eine dramatische Einschränkung des Nutzens, den dieses Konzept eigentlich mit sich bringt. Des Weiteren müsste der systemspezifische Teil des Konzepts für jedes auf dem Markt verfügbare ECAD-System neu implementiert bzw. angepasst werden.

3 Variantenkonstruktionsmodul

Das überarbeitete Konzept wird in drei einzelne Module gegliedert. Den Konfigurator, das Kopplungsmodul und das Interfacemodul. Die einzelnen Module werden mittels Datenaustausch in XML¹-Format miteinander kommunizieren. Eine Integration in ein bestehendes ECAD-System findet nicht statt, sondern es wird ausschließlich über die externen Schnittstellen eines angeschlossenen ECAD-Systems mit dem Interfacemodul kommuniziert.

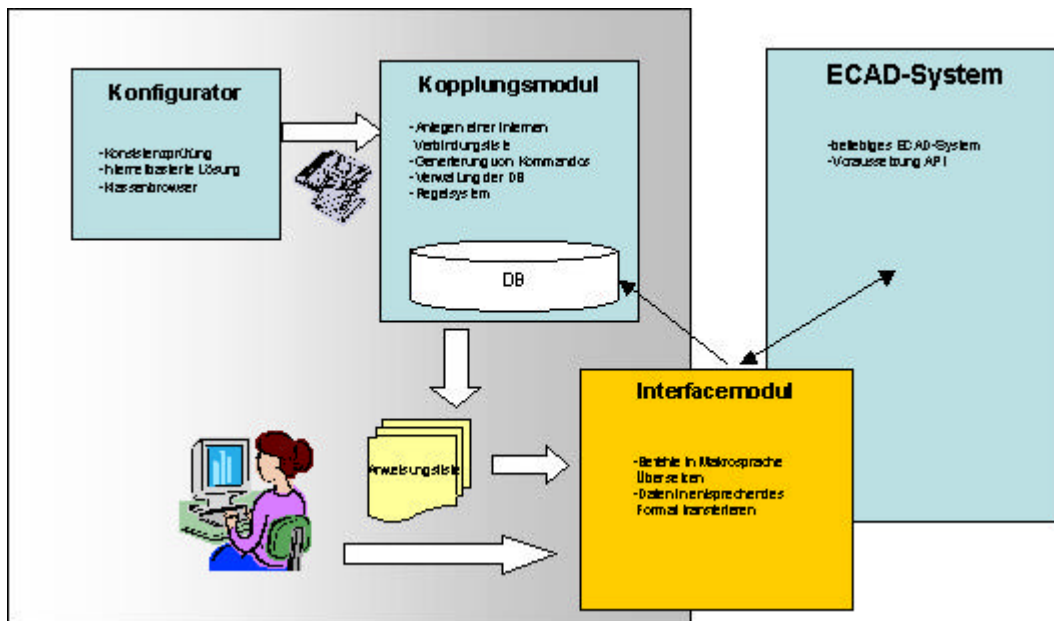


Bild (3) überarbeitetes Konzept

Der Arbeitszyklus wird in drei Phasen unterteilt.

In der ersten Phase wird die Datenbank des Kopplungsmoduls mit den Projekt- und Konstruktionsdaten aus dem angeschlossenen ECAD-System „gefüttert“.

In der zweiten Phase kann im Konfigurator ein neues Produkt oder aber auch nur eine neue Produktvariante konfiguriert werden.

In Phase drei erfolgt die Übergabe der Daten an ein ECAD-System, wo dann anhand der Daten ein neues Projekt erstellt wird.

Die Beschränkung auf die externen Schnittstellen eines ECAD-Systems wurde mit Bedacht gewählt. Eine Integration in ein ECAD-System ist faktisch nur dann realisierbar, wenn man Zugriff auf den Programmcode des ECAD-Kernels, bzw. die internen Schnittstellen des ECAD-Systems hat. Diese Hürde wäre eventuell noch mittels einer Industriekooperation mit einem Hersteller zu nehmen, ist aber sofort zum Scheitern verurteilt, wenn eine weitere Partei mit ins Boot aufgenommen werden soll. Weil jeder ECAD-Systemhersteller darauf bedacht ist, Dritten keinen Einblick in seine Technologien zu geben und nicht selber dazu beitragen will, seine Konkurrenten zu stärken. Eine selbstaufgelegte, oder aufgezwungene, Beschrän-

¹ XML: eXtensible Markup Language

kung auf einen Systemhersteller stellt aber wiederum eine erhebliche Einschränkung der Einsatzmöglichkeiten der Software dar. Außerdem verstößt diese Vorgehensweise gegen den grundsätzlichen wissenschaftlichen Ansatz, eine systemunabhängige Software zu entwickeln.

Eine Festlegung auf einen systemunabhängigen Ansatz mag auf den ersten Blick als der schlechtere Ansatz erscheinen, weil hier nicht aktiv auf ein ECAD-System und seine Funktionalitäten zurückgegriffen werden kann. So ist es beispielsweise hier nicht möglich, bei eventuell auftretenden Inkompatibilitäten von einzelnen Komponenten untereinander, sofort in das ECAD-System zu wechseln und dort dann „on-line“ zu versuchen, das Problem zu lösen. Um diese Problematik in den Griff zu bekommen, wäre es in unserem systemunabhängigen Ansatz erforderlich, in das Kopplungsmodul einen ECAD-Systemkern zu integrieren, der mindestens so mächtig an Funktionalität ist, wie das ECAD-System aus dem die Komponenten oder das Projekt importiert wurde. Dies sprengt aber ohne Zweifel die in diesem Softwareprojekt zur Verfügung stehende Zeit und Kosten. Des Weiteren würde dann nicht mehr ein reines Varianten- oder Konfigurationsmodul entwickelt, sondern ein eigenständiges und mächtiges ECAD-System.

Die Vorteile, die der systemunabhängige Ansatz mit sich bringt, liegen außer bei der offensichtlichen Eigenständigkeit des Produkts und bei den Synergieeffekten, die als „Abfallprodukt“ entstehen. So kann es als „Crosscompiler“ zur Konvertierung von Projekten vom ECAD-System A nach ECAD-System B genutzt werden. Es können Komponenten von Projekten, oder sogar ganze Projekte aus verschiedenen ECAD-Systemen zur Erstellung eines neuen Projekts verschmolzen werden, das dann in ECAD-System A und B oder einem beliebigen ECAD-System C zur Verfügung steht. Eine Voraussetzung hierfür ist natürlich, dass im Interfacemodul die entsprechenden Schnittstellen realisiert sind.

3.1 Die Datenstruktur

Als Datenstruktur für die Komponentendaten empfiehlt sich eine objektorientierte Speicherung in einer Klassenhierarchie (Baumstruktur). Diese ermöglicht auch eine intuitive Bedienung des Konfigurators mittels eines Klassenbrowsers. Der Klassenbrowser ist in einer Baumstruktur realisiert, so dass als Blätter in den Knoten des Baums die Instanzen/Varianten der Klasse aufgelistet werden. Die Knoten des Baums dürfen sowohl Blätter als auch weitere Subknoten haben. Zu den Instanzen werden nur die Parameterwerte gespeichert. Werden Komponenten oder Projekte aus einem ECAD-System importiert, das Parametrik unterstützt, so werden diese auch in der Klassenhierarchie als parametrische Komponenten verwaltet. Grundsätzlich werden in der Klassenhierarchie nur einzelne Komponenten und nicht ganze Projekte abgelegt. Die Verwaltung der Projekte wird in einer separaten Datenstruktur realisiert, von wo dann auf die zu einem Projekt gehörenden Instanzen von Komponenten in der Klassenhierarchie referenziert wird. Zur Realisierung der Datenstrukturen wäre es günstig, auf einen internetfähigen Standard, wie die XML zurückzugreifen. So kann auf diese Datenstrukturen auch vom Konfigurator, der in einem Internetbrowser realisiert werden soll, ohne allzu großen Aufwand zugegriffen werden.

Weil die Auswahl einer geeigneten Datenstruktur einen zentralen Teilaspekt des Projekts darstellt und den zur Verfügung stehenden zeitlichen Rahmen sprengen könnte, wäre es von Vorteil, wenn man auf eine schon bestehende Datenstruktur, die eventuell nur noch leicht angepasst werden muss, zurückgreifen könnte. Aus heutiger Sicht würde sich beispielsweise die Interface-Datenstruktur, die der Arbeitskreis „Integration der elektrischen und der mechanischen Konstruktion von Schaltschränken mit CAD-Systemen“ (kurz Schaltschrank-

Engineering), der GI¹ [8], im Rahmen des Projektes „EMface – eine herstellerneutrale Schnittstelle zwischen ECAD- und MCAD-Systemen“ entwickelt hat, als Arbeitsgrundlage anbieten. Dort ist schon ein Schema für Konstruktionsbestandteile unterschiedlicher Granularität definiert. Die EMFace-Datenstruktur ist mittels XML definiert.

3.2 Konfigurator

Der Konfigurator soll in einem Internetbrowser als Java-Applet lauffähig sein. Er bekommt seine Informationen, indem er eine Austauschdatei vom Kopplungsmodul einliest. Der Konfigurator soll dem Benutzer nach dem Start zunächst zwei Modi zur Verfügung stellen und dann in den Konfigurationsmodus wechseln. Im ersten der beiden Startmodi wird zuerst ein schon bestehendes Projekt geöffnet und dann verändert. D.h. der Benutzer kann basierend auf dem alten Projekt neue Produktvarianten konfigurieren, oder aber ein neues Projekt anlegen. Der andere Startmodus erlaubt es dem Benutzer mittels des Klassenbrowser auf die in der Klassenhierarchie vorhandenen Komponenteninstanzen zuzugreifen und anhand dieser Ausgangskomponente ein neues Produkt/Projekt zu konfigurieren.

Beschreibung der Betriebsmodi:

Modus 1 „Template laden“:

Es wird ein in der Datenstruktur ein schon vorhandenes Produkt als „Basiskonstruktion“ ausgewählt, um dann ausgehend von diesen Daten im Konfigurator eine Variante dieser Konstruktion oder aber ein neues Produkt zu konfigurieren. Nach der Auswahl der „Basiskonstruktion“ wird automatisch in den Komponentenkonfigurationsmodus gewechselt.

Modus 2 „Neukonfigurationsmodus“:

Erstellung eines neuen Produkts anhand von Einzelkomponenten und Modulen, die in der Datenbank hinterlegt sind. D.h. im Klassenbrowser wird die Instanz einer Klasse ausgewählt, die als Ausgangsbasis für ein neues Produkt dienen soll. Als Ausgangskomponente kann natürlich nur eine Klasse ausgewählt werden, für die auch eine Implementierung in der Moduldatenbank vorhanden ist. Danach wird in den Komponentenkonfigurationsmodus gewechselt.

Modus 3 „Komponentenkonfiguration“

Zur Basiskonstruktion bzw. Ausgangsbasis können hier selektiv weitere Instanzen von Komponenten mittels des Klassenbrowsers hinzugefügt werden. Sobald eine weitere Komponente ausgewählt ist, erfolgt eine Überprüfung auf Kompatibilität der externen Schnittstellen der beiden Komponenten. Sind die externen Schnittstellen kompatibel, d.h. die Schnittstellen sind identisch, oder die neu ausgewählte Komponente benötigt nur eine Untermenge der zur Verfügung stehenden Schnittstellen, so kann eine weitere Komponente hinzugefügt, oder aber das Projekt oder die neue Produktvariante dem Kopplungsmodul übergeben werden.

Möglichkeiten zur Realisierung des Konfigurators

Dank des systemunabhängigen Ansatz ist man jetzt in der Lage, jeden erdenklichen „Konfigurator“ an das Variantenmodul zu koppeln. Dieser muss lediglich ein Format liefern, das von dem Variantenmodul verstanden wird. Als Konfiguratoren können Anwendungsprogramme jeglicher Komplexität und Technologie verwendet werden, wie beispielsweise:

¹ GI: Gesellschaft für Informatik e.V.

- **MS Access[®]**
Die Nutzung vom Access bietet sich deshalb an, weil es sich zunächst einmal um ein Standardprogramm handelt, das überall zu einem relativ niedrigen Preis erhältlich ist. Des Weiteren ist Access bereits eine Datenbank, in der man die Daten des Konfigurators (Produktdaten, Komponentendaten) effizient verwalten kann. Reports oder Queries können mittels Access einfach generiert werden. Für den prototypischen Einsatz wäre Access auch deshalb geeignet, da man damit auch ohne viel Aufwand Benutzermasken anlegen kann.
- **MS Excel[®]**
Neben der Tatsache, dass Excel sehr weit verbreitet ist, bietet Excel eine schnelle Möglichkeit, Daten darzustellen und zu modellieren. Da der Preis von Excel verhältnismäßig niedrig und das Programm an und für sich ziemlich einfach zu verstehen ist, ist Excel mit Sicherheit eine Alternative bei der Auswahl der Konfiguratoren. Außerdem unterstützt Excel ab der Version 2002 auch XML, so dass eine weitere Konvertierung der Daten entfallen würde.
- **Komplexe konfigurierbare Konfiguratoren**
Diese Konfiguratoren, wie beispielsweise das kommerziell verfügbare Produkt SECON[®] von der Firma camos [9] würden sich für das Forschungsprojekt besonders gut eignen. Da aber der Preis, den man für sie ausgeben muss, zu hoch und somit nicht finanzierbar ist, können sie bei der Auswahl nicht berücksichtigt werden. Vorteile dieser Programme sind unter anderem, dass man sie auf seine Bedürfnisse beliebig konfigurieren kann, und sie sich auch oft so konfigurieren lassen, dass sie das gewünschte Format exportieren. Ein wesentlicher Nachteil ist aber auch, dass man für die Konfiguration eines solchen Programms einen Consultant¹ der Firma der Produktes benötigt, da die Einarbeitung aufwendig ist.
- **Eigene Programme**
Prinzipiell eignen sich natürlich eigene Programme für einen Einsatz als Konfigurator am besten. Hier ist es möglich, sowohl das Erscheinungsbild als auch das Format der zu exportierenden Daten selbst zu definieren. Leider dauert die Implementierung solch eines Konfigurators im Vergleich zu den anderen Möglichkeiten zu lange, als dass man sie für eine prototypische Implementierung einsetzen könnte.
- **Internet Technologie (HTML, Java,...)**
Eine internetbasierende Lösung besitzt einige Vorteile. Zunächst ist zu sagen, dass die Erstellung der Oberfläche recht einfach ist, da hier kommerzielle Programme, wie etwa „Frontpage[®]“ [10] oder „Dream Weaver[®]“ [11] eingesetzt werden können. Selbst der Einsatz von Freeware Programmen, wie z.B. der „Netscape Composer[®]“ [12], bietet eine für das Projekt genügend gute Lösung. Der Einsatz eines internetbasierenden Konfigurators erlaubt es dem Benutzer, den Konfigurator als ein lokales Programm oder auch als Intranet-/Internetdienst zu nutzen. Dies könnte zu einer Mehrbenutzerfähigkeit des Konfigurators erweitert werden. Eine Installation des Konfigurators auf einem Web-Server stellt die administrative Hauptaufgabe dar, da bei einem Intranet-/Interneteinsatz keine lokale Installation erfolgen muss. Im Falle von Java könnten

¹ Consultant: Berater

eventuell erforderliche jar¹ Files per Java Web Start auf die Clients verteilt werden. Bei jedem Start (konfigurierbar) der Applikation wird dann automatisch auf dem Server nach Updates geschaut, die ggf. heruntergeladen werden. Es gäbe hier also eine minimale Verwaltung und um die Verteilung kümmert sich Java Web Start. Die Betriebssysteme der heutigen Zeit besitzen einen integrierten Browser. Ein weiterer wichtiger Vorteil bei diesem Ansatz besteht darin, dass das XML-Format von den meisten gängigen Browsern unterstützt wird.

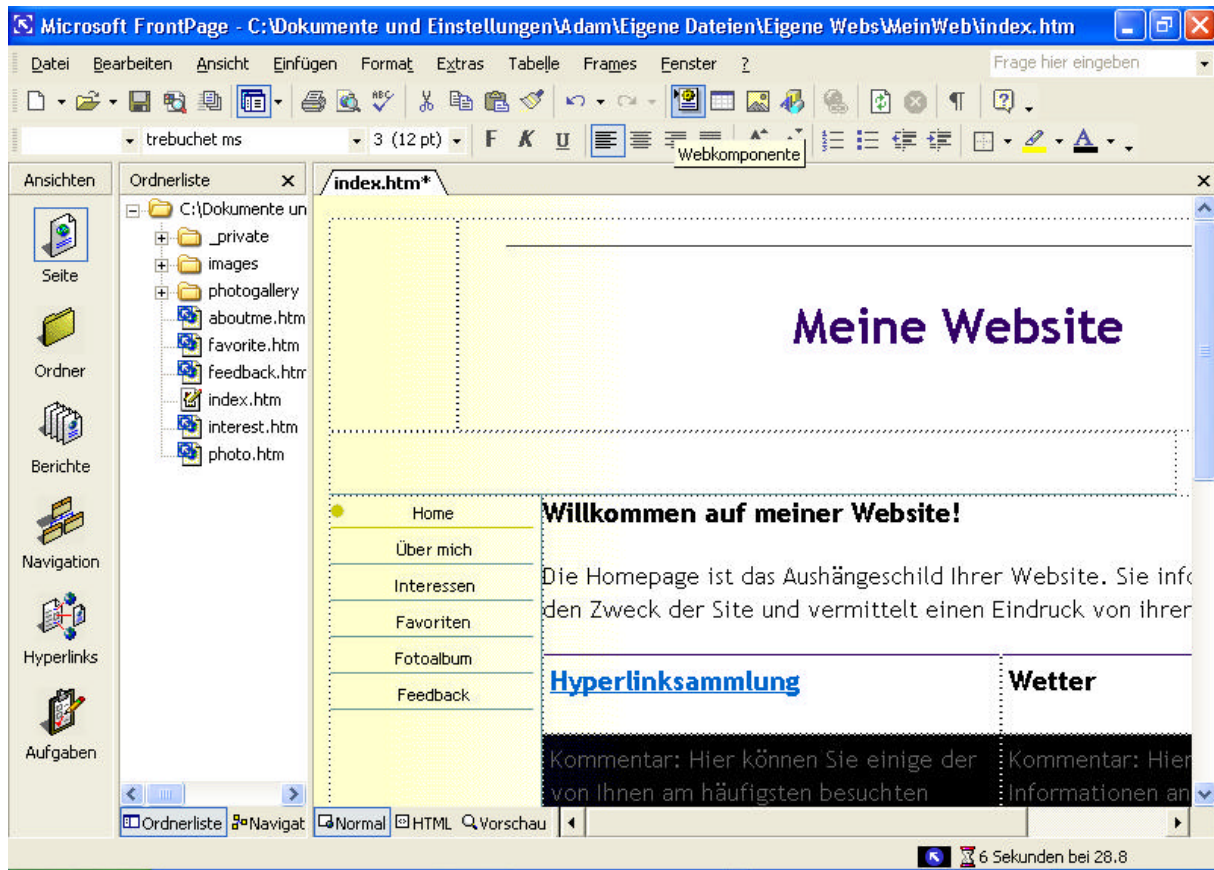


Bild (4) Microsoft FrontPage 2000, [10]

3.3 Kopplungsmodul

Im Kopplungsmodul erfolgt die Verwaltung aller Komponenten und Projektdaten, kurz des Produktwissens, das in einer Datenbank gespeichert ist. Das Kopplungsmodul soll als einziges Modul direkt auf eine Datenbank zugreifen können. D.h. der Konfigurator und das Interface erhalten Informationen nur, indem sie eine Anfrage an das Kopplungsmodul stellen. Dieses unterscheidet, von welchem Modul die Anfrage kam, und kann die entsprechenden Informationen weiterreichen.

- Zum einen, falls die Anfrage von Konfigurator kam, muss es Informationen über Bauteile liefern können. Ebenso sollte das Kopplungsmodul die Informationen über das Projekt liefern können, da man bereits im Konfigurator eine Auswahl, bzw. eine Vergabe des neuen Namens durchführen muss.

¹ jar: Java Archive, Dateieindung von Dateien in Java.

- Zum anderen muss das Kopplungsmodul sowohl die Produktdaten, als auch Anweisungen, die zum Generieren bzw. Ändern eines Projekts nötig sind, übergeben.

Die Anweisungen, die das Kopplungsmodul generieren soll, werden im Prinzip im Konfigurator zunächst durch den Benutzer selektiert. Hierbei kann dieser auswählen, ob ein neues Projekt angelegt werden soll, oder ob man in einem bestehenden Projekt Änderungen vornehmen will. Dabei ist zu beachten, dass es dazu notwendig ist, dem Konfigurator eine Liste mit bereits verfügbaren Projekten zur Verfügung zu stellen.

Das Kopplungsmodul beinhaltet das Konstruktionswissen zur Umsetzung der Konfigurationen in die elektrische Dokumentation. Das heißt, es stellt die Aktionen, die der Konstrukteur früher manuell im Elektro-CAD durchgeführt hätte, zu einer Sequenz von Anweisungen zusammen. Das Kopplungsmodul nimmt die Komponentenliste des Konfigurators entgegen und setzt diese über produktspezifische Regeln, die für die Komponentenkomposition hinterlegt sind, in Anweisungen um. Die Anweisungen können entweder in einer Datei abgelegt, oder aber an das Schnittstellenmodul weitergereicht werden. Sie müssen in einer geeigneten Sprache vorliegen (siehe Kapitel 5.2). Die Regeln müssen für jedes Produkt (bzw. jede Produktfamilie) in einer Consulting-Leistung oder durch den Ingenieur/Administrator (Wissensingenieur) beim Kunden ins System eingebracht werden. Bei diesen Regeln handelt es sich um Abhängigkeiten von Komponenten und Bildungsvorschriften, aus denen Anweisungen abgeleitet werden. Hier empfiehlt sich, kommerzielle Applikationen einzusetzen, da die Programmierung eines Konfigurators mit viel Aufwand verbunden ist.

Eine der weiteren Funktionalitäten, die das Kopplungsmodul zur Verfügung stellen muss, ist die Überprüfung bzw. Neuvergabe von Namen. Hierbei handelt es sich sowohl um Blattbezeichnungen, als auch Namen der Betriebsmittel. Diese müssen beim Generieren automatisch vergeben werden, ohne dass der Konstrukteur eingreifen müsste. Das größte Problem hierbei stellt allerdings die Umbenennung der Betriebsmittel dar. In diesem Fall müssen alle Querverweise angepasst werden. Mit einem sehr hohen Aufwand verbunden ist die Umbenennung in den freidefinierbaren Textfeldern. Hier müssen mit einer „Suche und Ersetze“ Strategie alle verfügbaren Textfelder durchsucht und gegebenenfalls die Namen geändert werden.

Neben der Namensvergabe muss das Kopplungsmodul die Betriebsmittelkennziffer (BMK) eindeutig bestimmen. Dabei werden doppelt vorkommende Kennziffern umbenannt, so dass die Daten, die an das ECAD-System übergeben werden, eindeutig sind. Dabei ist durchaus denkbar, dass das Kopplungsmodul die nächste verfügbare Zahl ermittelt und diese dann als BMK dem Bauteil zuweist.

Das Kopplungsmodul ist in der Lage, nach einer Prüfung der Schaltung eine interne Liste anzulegen, in welcher die einzelnen Verbindungen gespeichert sind. Diese stellt im Prinzip das Herzstück der Schaltung dar, da aus dieser die Schaltung generiert werden kann. Dabei muss das Kopplungsmodul die Zuweisung der Anschlüsse automatisch durchführen, und bei Inkonsistenzen eine Interaktion vom Konstrukteur abwarten.

3.4 Interface

Im Interfacemodul wird die Kommunikation mit einem ECAD-System realisiert. Hier muss für jedes ECAD-System, mit dem kommuniziert werden soll, je ein Schnittstellensatz von zwei Konvertern realisiert werden. Der erste Konverter wird benötigt, um Informationen aus dem ECAD-System in die Metasprache die Datenstruktur zu konvertieren. Der zweite Konverter muss, je nach ECAD-System, genau die Umkehrfunktion des ersten Konverters reali-

sieren, oder aber eine Umsetzung der Metasprache in die Makrosprache des anzusteuernenden ECAD-Systems, um dort dann das neue Projekt oder die Produktvariante zu konstruieren. Während der Ansteuerung des ECAD-Systems zum Erstellen einer neuen Konstruktion sollte eine bidirektionale Kommunikation zwischen dem ECAD-System und dem Interface stattfinden, um auf eventuell auftretende Fehlermeldungen reagieren zu können.

3.5 Vorgehensweisen bei der Realisierung

Grundsätzlich könnte jetzt sofort mit der Realisierung der einzelnen Module, in einer als geeignet erachteten Programmiersprache, begonnen werden. Bei dieser Vorgehensweise kommt man dann aber auch ziemlich schnell an einen Punkt, wo sich neue noch nicht bedachte Probleme auftun, die sich dann mit dem bisher erarbeiteten Konzept oder Programmdesign nicht mehr bewältigen lassen, und im schlimmsten Fall einen kompletten Neuanfang erfordern. Deshalb ist es unerlässlich, sich vor dem Beginn einer Implementierung mit den notwendigen Grundlagen auseinander zusetzen, und anhand der dort zur Verfügung gestellten Hilfsmittel eine strukturierte Vorgehensweise zu planen.

Zusätzlich sollten folgende Aspekte bei der Realisierung nicht aus den Augen verloren werden:

- **Nutzung von bestehenden Ressourcen:** existierende Datenbestände können eingebunden werden, vorhandene Techniken und Erfahrungen können genutzt werden.
- **Transparente Entwicklung:** Um verschiedene Betriebssysteme und ECAD-Systeme unterstützen zu können, muss ein systemunabhängiges Kommunikationskonzept entworfen werden.

4 Grundlagen der Informatik

Im folgenden Kapitel werden kurz die unbedingt notwendigen Grundlagen zu Grammatiken, Compilerbau und Software-Engineering vermittelt, die zwingend notwendig sind, um mit einer erfolgreichen Realisierung eines Variantenmoduls beginnen zu können. Auf eine detaillierte Begründung, für welche Realisierungsaspekte genau die hier betrachteten Grundlagen benötigt werden, wird an dieser Stelle verzichtet. Sie werden im Laufe der weiteren Kapitel offensichtlich.

4.1 Grammatiken

Grammatiken sind Regelsysteme, mit denen sich genau die Wörter, die zu einer Sprache gehören, erzeugen lassen. Für Programmiersprachen bedeutet das, es sind Regeln, mit deren Hilfe sich syntaktisch korrekte Programme erstellen lassen.

Grammatiken wurden formal durch den Linguisten Noam Chomsky [13] definiert, und bestehen aus vier Komponenten :

1. Den Terminalzeichen T (oder Σ), einem endlichen Alphabet, über dem die Sprache definiert ist.
2. Den Variablen V , einer endlichen, zu T disjunkten Menge von Hilfszeichen.
3. Einem Startsymbol $S \in V$.
4. Einer endlichen Menge von Ableitungsregeln (oder Produktionsregeln) P . Ableitungsregeln bestehen aus einer linken und einer rechten Seite, also einem Paar (l,r) . Für die linke bzw. rechte Seite einer Regel muss folgendes gelten $l \in (V \cup T)^+$ und $r \in (V \cup T)^*$. Eine Ableitungsregel kann dann wie folgt eingesetzt werden: Kommt in einem Wort z die linke Seite einer Ableitungsregel P vor, so darf das Vorkommen der linken Seite von P durch die rechte Seite von P ersetzt werden.

Definition Linksableitung

Unter einer Linksableitung versteht man eine Ableitung, in der in jedem String¹ die Variable, die am weitesten links steht, abgeleitet wird.

Definition Rechtsableitung

Unter einer Rechtsableitung versteht man eine Ableitung, in der in jedem String die Variable, die am weitesten rechts steht, abgeleitet wird.

¹ String: unter einem String wird in diesem Kontext eine Aneinanderreihung von Terminalsymbolen und Variablen verstanden.

Definition der Ableitungsnotation

Für die Strings w, z gilt:

$w, z \in (V \cup T)$

- Die Notation $w \stackrel{\circ}{\rightarrow} z$ bedeutet, dass sich z in einem Schritt durch Anwendung einer Produktion, mittels einer beliebigen Ableitung, aus w ableiten lässt.
- Die Notation $w \stackrel{*}{\rightarrow} z$ bedeutet, dass sich z aus w in endlich vielen Schritten, mittels einer beliebigen Ableitung, ableiten lässt.
- Die Notation $w \stackrel{r}{\rightarrow} z$ bedeutet, dass sich z in einem Schritt durch Anwendung einer Produktion, mittels einer Rechtsableitung, aus w ableiten lässt.
- Die Notation $w \stackrel{r*}{\rightarrow} z$ bedeutet, dass sich z aus w in endlich vielen Schritten, mittels einer Rechtsableitung, ableiten lässt.

Definition der vier Grammatikklassen der Chomsky-Hierarchie:

- Grammatiken ohne Einschränkungen heißen Grammatiken vom Typ Chomsky-0
- Grammatiken, bei denen alle Produktionen der Form $u \rightarrow v$ mit $u \in V^+$, $v \in ((V \cup T) - \{S\})^+$ und $|u| \leq |v|$ oder $S \rightarrow \varepsilon$ sind, heißen kontextsensitiv oder Grammatiken vom Typ Chomsky-1
- Grammatiken, bei denen alle Produktionen der Form $A \rightarrow v$ mit $A \in V$ und $v \in (V \cup T)^*$ sind, heißen kontextfrei oder Grammatiken vom Typ Chomsky-2
- Grammatiken, bei denen alle Produktionen der Form $A \rightarrow v$ mit $A \in V$ und $v = \varepsilon$ oder $v = aB$ mit $a \in T$ und $B \in V$ sind, heißen rechtslinear, regulär oder Grammatiken vom Typ Chomsky-3

Weitere Informationen in [13]

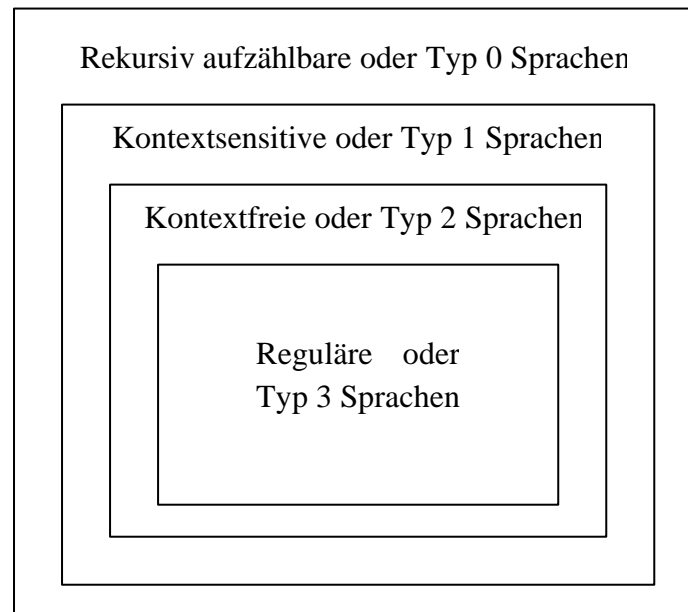


Bild (5) Anordnung der Sprachen der Chomsky-Hierarchie nach ihrer Mächtigkeit. In Anlehnung an [14]

Für den Compilerbau, auf den später in diesem Kapitel noch näher eingegangen wird, sind folgende Subtypen vom Typ Chomsky-2 von Relevanz: LL, simple LL (SLR), LR, simple LR (SLR) und lookahead LR (LALR).

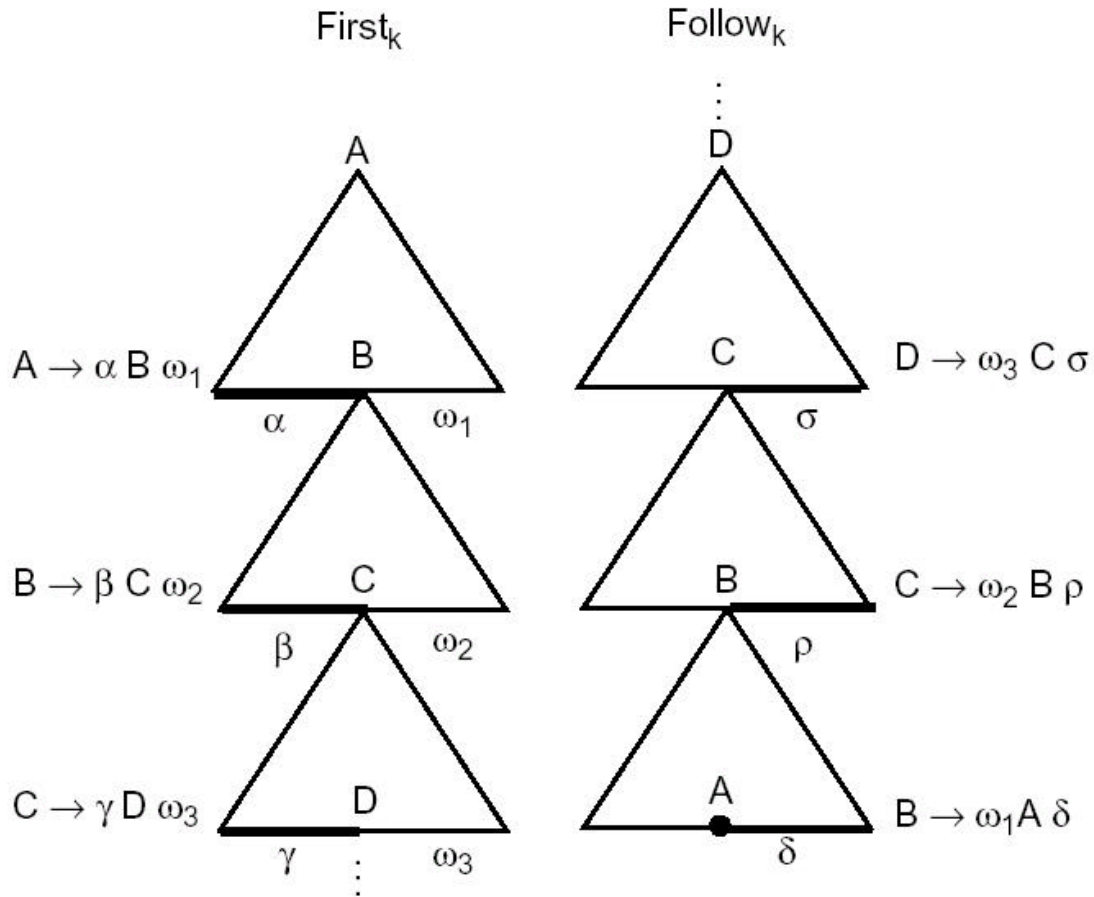
Der erste Buchstabe bei den LL und LR steht dafür, dass die Eingabe von links nach rechts gelesen wird. Der zweite Buchstabe zeigt an, ob es sich um Linksableitungen oder Rechtsableitungen handelt.

Um diese Grammatiken zu definieren, werden noch so genannte FIRST- und FOLLOW-Mengen benötigt. Die Definitionen und die Algorithmen der FIRST- und FOLLOW-Mengen erfolgen nach [15].

Definition FIRST- und FOLLOW-Mengen:

- $FIRST_k(\alpha) = \{k:w \mid \alpha \rightarrow^* w, w \in T^*\}$ d.h. die Menge aller Terminalzeichenfolgen w mit Länge k , die ein von α ableitbares Wort der Grammatik einleiten.
- $FOLLOW_k(A) = \{k:w \mid S \rightarrow^* \alpha A \beta, w \in FIRST_k(\beta)\}$ d.h. die Menge aller Terminalzeichenfolgen w mit Länge k , die in einer Ableitung von S unmittelbar auf A folgen können

Veranschaulichung der First_k - und Follow_k -Mengen



$\text{First}_k (A) =$

$\text{First}_k(\alpha\beta\gamma\dots\omega_3\omega_2\omega_1)$

lokale k -Folgemenge von $A =$

$\text{First}_k(\delta\rho\sigma\dots)$

Bild (6) Veranschaulichung der First_k - und Follow_k -Mengen nach [15]

$\text{Follow}_k(A) =$ Vereinigung der lokalen k -Folgemengen von A für alle Vorkommen von A in Ableitungen $S \rightarrow^* |Ar$

Berechnung der First₁-Mengen

Regeln:

1. $\forall t \in T: \text{FIRST}(t) = \{t\}$
2. $\forall p \in P: X \rightarrow \varepsilon : \varepsilon \in \text{FIRST}(X)$
3. $\forall p \in P: X \rightarrow Y_1 Y_2 \dots Y_n, Y_i \in T \cup V:$
 - a. wenn $a \in \text{FIRST}(Y_i)$, $a \in T$, und $\forall j, j < i: \varepsilon \in \text{FIRST}(Y_j)$, dann $a \in \text{FIRST}(X)$
 - b. wenn $\forall i, i=1 \dots n: \varepsilon \in \text{FIRST}(Y_i)$, dann $\varepsilon \in \text{FIRST}(X)$

Algorithmus zur Berechnung durch iterative Erweiterung der First-Mengen

- wende Regeln 1 und 2 an
- **repeat**
 für alle $p \in P$, wende Regel 3 an
 until die innere Schleife bewirkt keine Veränderungen der FIRST-Mengen mehr.

und schließlich:

$\text{FIRST}(\alpha)$, $\alpha = Y_1 Y_2 \dots Y_n$, wird nach 3.a und 3.b entsprechend bestimmt.

Berechnung der Follow₁-Mengen

Regeln:

1. für Startsymbol $S_G: \$ \in \text{FOLLOW}(S_G)$ ($\$$ = "Ende der Eingabe"-Token)
2. $\forall p: X \rightarrow \alpha? \gamma : (\text{FIRST}(\gamma) - \varepsilon) \subset \text{FOLLOW}(B)$
3. $\forall p: X \rightarrow \alpha B$ oder $\alpha B \gamma$ und $\varepsilon \in \text{FIRST}(\gamma)$
 $\text{FOLLOW}(X) \subset \text{FOLLOW}(B)$

Algorithmus zur iterativen Berechnung nach obigen Regeln

1. wende Regeln 1 und 2 an
2. **repeat**
 für alle $p \in P$, wende Regel 3 an
 until die innere Schleife bewirkt keine Veränderungen der Follow-Mengen mehr.

LL(1) Grammatiken

Eine Grammatik ist vom Typ LL(1), wenn folgende Bedingungen erfüllt sind:

für alle $A, A \in V$ und für alle $p_1, p_2 \in P, p_1: A \rightarrow \alpha; p_2: A \rightarrow \beta$, gilt:

1. falls $\alpha \xrightarrow{*} t_1\gamma, \beta \xrightarrow{*} t_2\delta, t_1, t_2 \in T$ dann $t_1 \neq t_2$, d.h. $FIRST_1(\alpha) \cap FIRST_1(\beta) = \emptyset$
2. es gilt höchstens eine der Ableitungen $\alpha \xrightarrow{*} \varepsilon, \beta \xrightarrow{*} \varepsilon$
3. falls $\beta \xrightarrow{*} \varepsilon, \alpha \xrightarrow{*} t_1\gamma$, dann $t_1 \notin FOLLOW_1(A)$

Simple LL(1) (SLL(1) Grammatiken

Eine Grammatik ist vom Typ SLL(1), wenn folgende Bedingungen erfüllt sind:

1. keine ε -Produktionen¹ hat, und
2. für alle $A, A \in V$ und für alle $p_1, p_2 \in P, p_1: A \rightarrow \alpha; p_2: A \rightarrow \beta$, gilt:
 $\alpha = t_1\gamma; \beta = t_2\delta; t_1 \neq t_2; t_1, t_2 \in T$

LR(k) Grammatiken

Eine Grammatik ist vom Typ LR(k), wenn folgende Bedingungen erfüllt sind:

1. $S \xrightarrow{r^*} \alpha X w \xrightarrow{r} \alpha \beta w$
2. $S \xrightarrow{r^*} \gamma Y X \xrightarrow{r} \alpha \beta y$
3. $FIRST_k(w) = FIRST_k(y)$

Und aus 1-3 folgt $\alpha = \gamma, X = Y$ und $y = x$

¹ ε -Produktion: Ableitung einer Variable oder eines Strings zum leeren Wort

4.2 Grundlagen des Compilerbaus

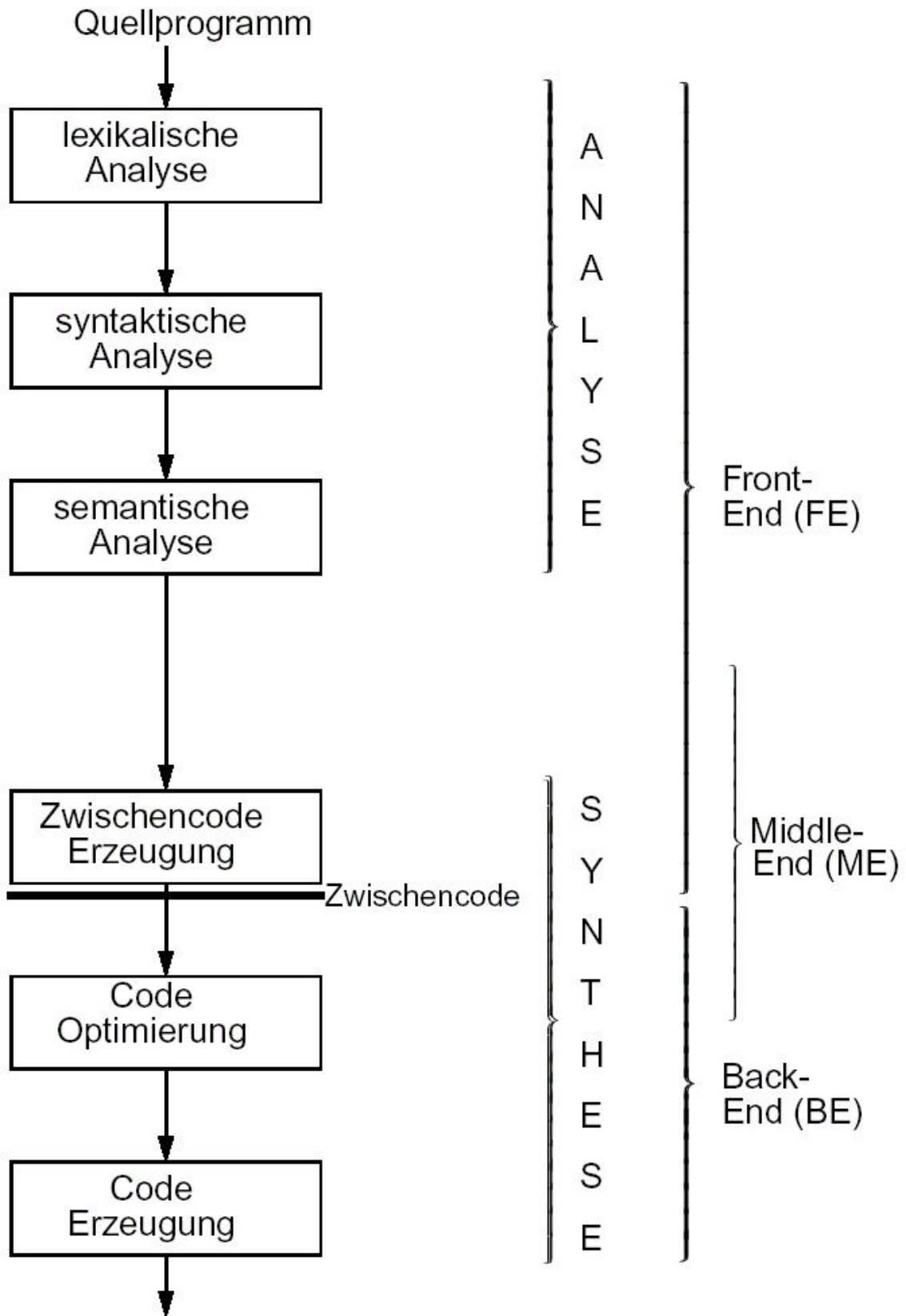


Bild (7) Feinmodell eines Compilers nach [15]

Das ganze Kapitel 4.2 orientiert sich an [15], [16].

Im Rahmen des Softwareentwicklungsprozesses zur Erstellung eines Variantenmoduls wird man spätestens bei der Implementierung des Interfaces mit Verfahren und Techniken die zum Basiswissen des Compilerbaus gehören, konfrontiert werden. Deshalb wird im folgenden eine kurze Einführung in die Arbeitsweise und die Techniken des Compilerbaus gegeben.

Front-Ends für verschiedene Programmiersprachen

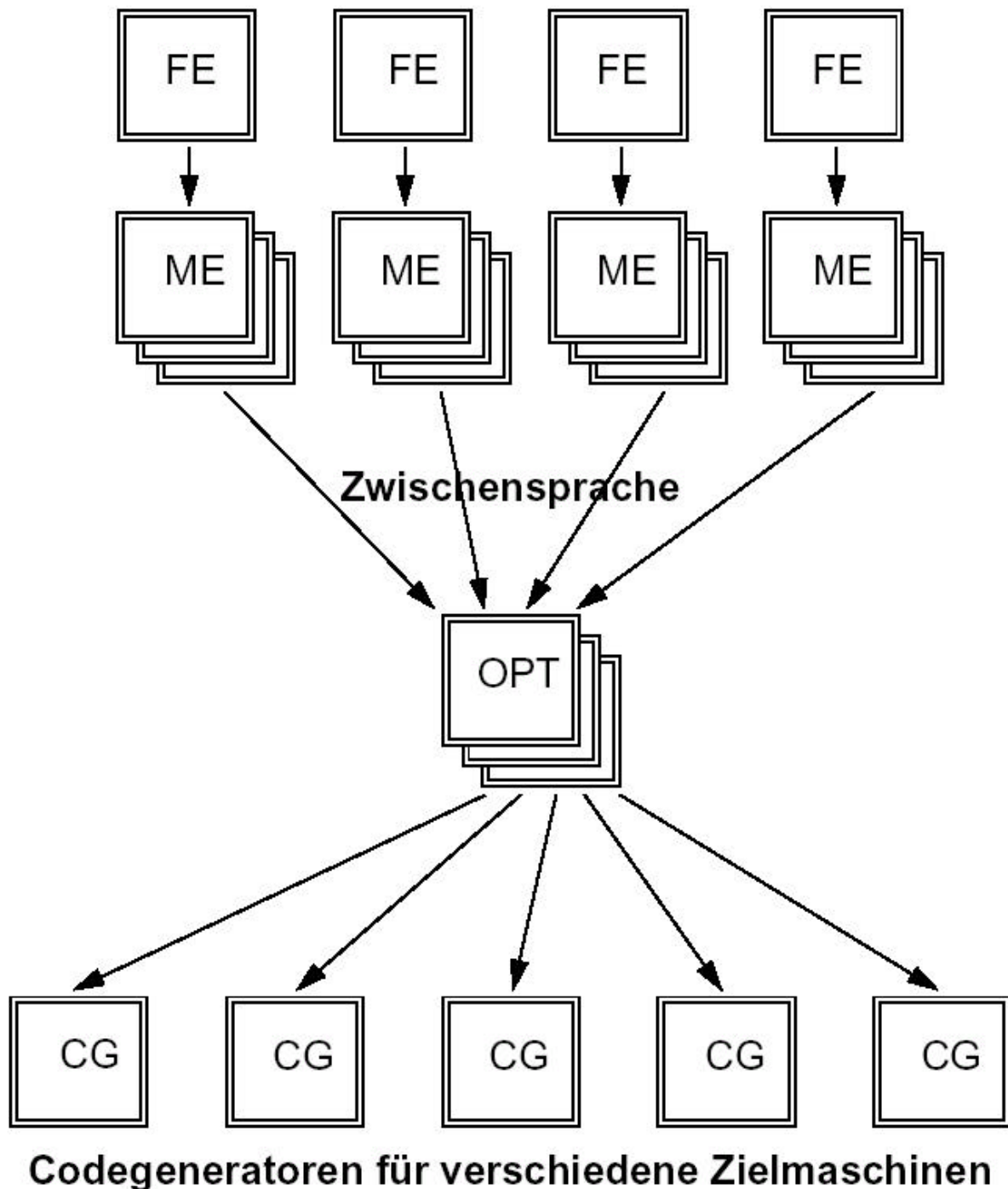


Bild (8) Übersetzer Familie nach [15]

Einfach ausgedrückt liest ein Compiler ein Programm, das in einer Sprache, der Quellsprache, geschrieben ist ein, und übersetzt es dann in ein äquivalentes Programm in einer anderen Sprache, der Zielsprache. Ein wichtiger Aspekt dieses Übersetzungsvorgangs ist, dass ein Compiler den Benutzer über eventuell vorhandene Fehler im Quellprogramm informiert und warnt.

Grundsätzlich wird die Übersetzung eines Programms in zwei große Teilschritte aufgeteilt, nämlich die Analyse und die Synthese.

Der Compiler selbst ist in ein Front-End (FE), Middle-End (ME) und Back-End (BE) unterteilt. Zum FE werden die komplette Analyse- und frühe Synthesephasen gerechnet, die weitgehend abhängig von der Quellsprache und unabhängig von der Zielsprache sind. Zum BE werden die Synthesephasen gerechnet die abhängig von der Zielsprache/Zielmaschine sind. Im ME ist der Rest des Compilers zusammengefasst, der sich weder dem FE noch dem BE zuordnen lässt. Die Einteilung in FE, ME und BE erfolgt unter dem Aspekt der Anpassbarkeit und Wartbarkeit des Compilers. Soll beispielsweise der Compiler nicht mehr Sprache A nach Sprache B übersetzen, sondern Sprache C jetzt nach Sprache B, so müssen die Compilerbauer nur das FE des Compilers, das abhängig von der Quellsprache ist, auf Sprache C anpassen. Analog hierzu verhält es sich mit dem BE, nur dass hier an die Zielsprache angepasst wird.

4.2.1 Analyse

Bei der Analyse des Quellprogramms werden die lexikalische, syntaktische und semantische Analyse durchgeführt. Die Analyse wird aus mehreren Gründen in einzelne Unterphasen unterteilt:

Ein einfacheres Design des Compilers ist wohl der wichtigste Grund. Eine Trennung der Lexikalische Analyse von der Syntaktischen Analyse erlaubt es, eine der beiden Phasen zu vereinfachen. So ist beispielsweise ein Parser¹, der auch noch Regeln für Kommentare und Leerzeichen enthält, wesentlich komplexer, als ein Parser, der davon ausgehen kann, dass sich um diese Dinge bei der Lexikalische Analyse gekümmert wurde.

Höhere Effizienz des Compilers. Ein separater Lexer² erlaubt eine bessere und effizientere Verarbeitung der Aufgaben. Ein großer Teil der Laufzeit wird durch das Lesen des Quellprogramms und der Aufteilung in einzelne Tokens verbraucht. Spezielle Puffertechniken für das Lesen von Eingabezeichen und die Verarbeitung von Tokens können die Gesamtperformance des Compilers erheblich verbessern.

Die Portabilität des Compilers wird erhöht. Eingabealphabet, Eigenheiten und andere Programm Besonderheiten, wie bspw. die Repräsentation von Spezial- oder Sonderzeichen, können so im Lexer gehalten werden.

Lexikalische Analyse

Die lexikalische Analyse ist die erste Arbeitsphase eines Compilers und wird durch den Lexer durchgeführt. Hier werden die Eingabezeichen eingelesen und daraus eine Folge von Tokens für den Parser erstellt, der die syntaktische Analyse durchführt. Wegen dieser engen Zusammenarbeit vom Lexer und Parser wird der Lexer üblicherweise als Unterprogramm des Par-

¹ Parser: der Parser ist eine Komponente eines Compilers und untersucht ob ein String von Tokens durch die ihm zu Grunde liegende Grammatik generiert werden kann

² Lexer: die Komponente des Compilers, welche die lexikalische Analyse durchführt.

sers realisiert. Wenn der Lexer das Kommando „get next token“ vom Parser bekommt, liest er die Eingabezeichen vom Quellprogramm, bis er das nächste Token erkennen kann. Weil der Lexer der Programmteil des Compilers ist, der den unveränderten Quellcode liest, führt er nebenbei auch noch eine Reihe weiterer Aufgaben durch. Eine dieser Aufgaben ist das Entfernen von Zeichen im Programmcode, die nur die Lesbarkeit des Programms erhöhen sollen, wie Leerzeichen, Tabulatoren, Kommentare und Zeilenumbruchszeichen. Eine weitere Aufgabe ist die Zuordnung von Fehlermeldungen des Compilers zu ihren Ursprüngen im Quellcode. So kann der Lexer sich beispielsweise die Anzahl der Zeilenumbruchzeichen, die er „gesehen“ hat merken, und dann zu einer Fehlermeldung die zugehörige Zeilennummer angeben.

Manchmal wird die Lexikalische Analyse nochmals in zwei aufeinander abfolgende Phasen unterteilt, dem Scanning und der Lexikalische Analyse. Hier ist der Scanner dann für die einfachen Aufgaben, wie das entfernen von Leerzeichen zuständig, während der Lexer die anspruchsvolleren Aufgaben übernimmt.

Syntaktische Analyse

Die Syntaktische Analyse wird durch den Parser durchgeführt. Der Parser bekommt eine Kette von Tokens und überprüft, ob die Zeichenfolge durch die Grammatik, die der Quellsprache zu Grunde liegt, abgeleitet wird. Vom Parser wird erwartet, dass er syntaktische Fehler auf eine verständliche Weise meldet. Außerdem sollte er einfache, oft vorkommende Fehler, automatisch temporär korrigieren und mit der Verarbeitung der Eingabe dann fortfahren.

Es existieren drei Haupttypen von Parsern. Universelle Parsingalgorithmen wie der CYK¹-Algorithmus und Earlys-Algorithmus können jede Grammatik parsen. Diese Parsingalgorithmen sind allerdings zu ineffizient um in nichtwissenschaftlichen Compilern eingesetzt zu werden. Die meist verwendeten Compiler gehören zu der Klasse der top-down oder aber der Klasse der bottom-up Compiler. Wie ihr Name schon verrät, bauen top-down Parser ihren Parsebaum ausgehend von der Wurzel bis in die Blätter ganz unten im Baum, während bottom-up Parser unten bei den Blättern beginnen und sich dann zur Wurzel hocharbeiten. Bei beiden Typen wird die Eingabe von links nach rechts und nur symbolweise gelesen.

Die leistungsstärksten top-down und bottom-up Parsingmethoden funktionieren aber nur auf Untermengen von Grammatiken. Einige dieser Unterklassen, wie die LL und LR Grammatiken, sind aber noch mächtig genug, um die meisten Syntaktischen Konstrukte, die in Programmiersprachen vorkommen, zu beschreiben. Parser, die von Hand geschrieben werden, benutzen oft die LL-Grammatiken. Parser die automatisiert erstellt werden, benutzen hingegen eher die größere Klasse der LR-Grammatiken.

Semantische Analyse

Die semantische Analysephase prüft das Quellprogramm auf semantische Fehler und sammelt Datentypinformationen, die in der später folgenden Codegenerierungsphase benötigt werden. Der Parsebaum, der in der Syntaktischen Analyse erstellt wurde, wird zur Identifikation der Operanden und Operationen von Ausdrücken und Befehlen benutzt. Ein wichtiger Teil der Semantischen Analyse ist die Datentypprüfung. Hier prüft der Compiler, dass jeder Operator nur Operanden hat, die auch in der Spezifikation der Quellsprache definiert sind. Teilweise muss der Compiler dann Datentypen konvertieren.

¹ CYK: Cocke-Younger-Kasami

4.2.2 Synthese

Die Synthese ist auch in drei einzelne Phasen unterteilt, nämlich die Zwischencodenerzeugung, die Codeoptimierung und die Codeerzeugung. Die Zwischencodenerzeugung geschieht im so genannten Middle-End des Compilers, die beiden anderen Phasen gehören zum Back-End.

Zwischencodenerzeugung

Nach der Syntaktischen und Semantischen Analyse generieren die meisten Compiler einen Zwischencode des Quellprogramms. Diesen Zwischencode kann man sich als Programm für eine abstrakte Maschine vorstellen. Der Zwischencode sollte unbedingt zwei Eigenschaften haben. Er sollte erstens leicht generierbar sein, und zweitens sich leicht in ein Zielprogramm übersetzen lassen.

Der Zwischencode kann also eine Vielzahl von verschiedenen Darstellungen haben, solange er nur die zwei oben genannten Eigenschaften besitzt. Eine Form von Zwischencode ist beispielsweise der Drei-Adress-Code, der eine Anlehnung an die Assemblersprachen ist. Im Drei-Adress-Code besteht ein Programm aus einer Folge von Anweisungen, die höchstens drei Operanden haben dürfen. Aufgrund dieser Beschränkung, kann man davon ausgehen, dass jede Anweisung nur eine Operation und eine Zuweisung hat. Der Compiler muss festlegen, in welcher Reihenfolge die Operationen durchgeführt werden. Er muss Hilfsvariablen generieren um die berechneten Ergebnisse zu speichern. Allgemein ausgedrückt müssen hier nicht nur Anweisungen berechnet werden, sondern auch Flusskontrollenkonstrukte und Prozeduraufrufe verarbeitet werden.

Codeoptimierung

Idealerweise sollten Compiler Code in der Zielsprache produzieren, der qualitativ gleichwertig ist mit Code, der von Hand erstellt wurde. Tatsächlich wird dieses Ziel aber nur sehr selten und mit großem Aufwand erreicht. Allerdings kann Code, der von einem nicht optimierenden Compiler erstellt wird, meist noch verbessert werden, so dass er schneller ausgeführt oder kompakter gespeichert wird, oder im Idealfall auch beides. Diese Verbesserungen in der Qualität des Programmcodes wird durch sogenannte Codeoptimierung erreicht. Der Begriff Codeoptimierung ist an sich nicht zutreffend, weil so gut wie nie garantiert wird, dass der erzeugte Code optimal, d.h. der bestmögliche Code, ist. Compiler, die eine Codeoptimierung durchführen, werden auch optimierende Compiler genannt.

Bei den Optimierungen, die vorgenommen werden können, kann man grundsätzlich zwei Arten von Optimierungen unterscheiden. Es gibt die maschinenunabhängige Optimierung, die den Zielcode verbessert, ohne auf zielmaschinenspezifische Eigenschaften zurückzugreifen. Und es gibt die maschinenabhängige Optimierung, die beispielsweise auf eine effiziente Registerbelegung achtet oder spezielle Maschinensprache Befehle ausnutzt.

Am geschicktesten ist es, zuerst die am meisten verwendeten Programmteile zu identifizieren, und diese Programmteile dann so effizient wie nur möglich zu machen. Eine zu diesem Vorgehen passende Aussage ist, dass 90% der Laufzeit eines Programms in 10% des Programmcodes verbraucht wird.

Mathematisch ist das Problem, optimalen Code zu generieren, unentscheidbar. Deshalb müssen wir uns in der Praxis mit heuristischen Techniken die „guten“ Code, wenn auch nicht optimalen Code generieren, zufrieden geben.

Codegenerierung

Die Codegenerierung ist der letzte Arbeitsschritt eines Compilers. Sie nimmt als Eingabe entweder den Zwischencode oder aber, wenn es ein optimierender Compiler ist, den optimierten Code des Quellprogramms, und erstellt daraus dann ein äquivalentes Programm in der Zielsprache. Bei der Codegenerierung muss jetzt auf die Eigenschaften der Zielsprache eingegangen werden, also Dinge wie Speicherbelegung, Speicherreservierung, Befehlsauswahl und Auswertungsreihenfolge. Das wichtigste bei der Codegenerierung ist allerdings, dass korrekter Code erzeugt wird.

4.3 Software Engineering

Das ganze Kapitel 4.3 orientiert sich an [17].

Software Engineering ist vereinfacht gesagt die Anwendung von systematischen Vorgehensweisen um fehlerfreiere, leichter wartbare und qualitativ hochwertigere Software mit möglichst niedrigen Entwicklungskosten zu erstellen. Motivation hierzu ist, dass die heutigen Systeme immer komplexer werden, und sich somit auch immer größere Software-Probleme ergeben.

Software Engineering wird u.a. wie folgt definiert:

- „Software Engineering ist die Entdeckung und Anwendung solider Ingenieur-Prinzipien mit dem Ziel, auf wirtschaftliche Art Software zu bekommen, die zuverlässig ist und auf realen Rechnern läuft.“ F.L. Bauer
- „Software Engineering ist die Herstellung und Anwendung einer Software, wobei mehrere Personen beteiligt sind oder mehrere Versionen entstehen.“ D.L. Parnas
- „Software Engineering
 - (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
 - (2) The study of approaches as in (1).“

IEEE Std. 610.12 [18]

4.3.1 Ziele des Software Engineerings

Das Hauptziel, die Senkung der Gesamtkosten, lässt sich in organisationsbezogene und aspektbezogene Ziele unterteilen.

Organisationsbezogene Ziele sind:

- Das Projektziel mit den geplanten Mitteln wie Personal und Budget, innerhalb der festgesetzten Termine und mit der geforderten Qualität erfolgreich zu beenden.
- Das Unternehmensziel, nämlich die Verbesserung der Ausgangsbasis für weitere Projekte durch Know-how Gewinn und Entwicklung wiederverwendbarer Software, zu erreichen.

Aspektbezogene Ziele:

- Produktivitätserhöhung
- Qualitätssteigerung
- Gebrauchsqualität der Software
- Wartungsqualität der Software
- Prozess-Qualität des Projekts
- Investitionsschutz für bestehende und entstehende Software

4.3.2 Der Software-Life-Cycle

Bei der Erstellung eines Produkts wird immer nach einer bestimmten Abfolge, wenn auch nicht bewusst, vorgegangen. So steht am Anfang die Idee oder der Wunsch, der dann konkretisiert wird. Dann wird eine Struktur festgelegt und anschließend zur Realisierung übergegangen. Ist das Produkt dann erstellt, so folgt eine Überprüfung der Qualität, die gegebenenfalls zu einer Änderung führt, oder aber der Auslieferung an den Kunden. Wenn das Produkt dann beim Kunden ist, können dort wiederum Änderungswünsche oder Reparaturen anfallen. Diese Abfolge von Schritten nennt man neudeutsch Life-Cycle oder im Fall von Software den Software-Life-Cycle.

Definitionen aus IEEE Std. 610.12, 1990

- **Cycle**
A period of time during which a set of events is completed.
- **Software life cycle**
The period of time that begins when a software product is conceived and ends when

the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and check phase, operation and maintenance phase, and, sometimes, retirement phase.

Note: These phases may overlap or be performed iteratively

- **Software development cycle**

The period of time that begins with the decision to develop a software product and ends when the software is delivered. This cycle typically includes a requirements phase, test phase, and sometimes, installation and checkout phase.

Notes: (1) The phases listed above may overlap or be performed iteratively, depending upon the software development approach used. (2) This term is sometimes used to mean a longer period of time, either the period that ends when the software is no longer being enhanced by the developer, or the entire software life cycle.

- **System life cycle**

The period of time that begins when a system is conceived and ends when the system is no longer available for use.

Weitere Informationen zu IEEE Std. 610.12, 1990 [18]

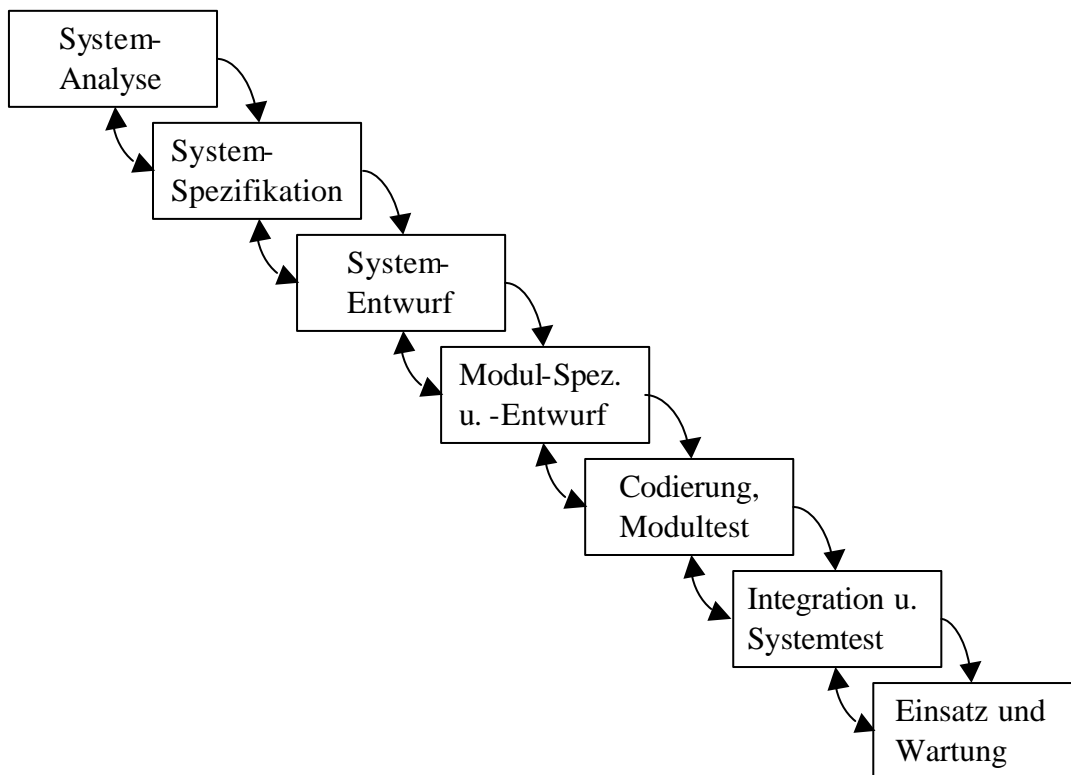


Bild (9) Das Wasserfallmodell nach Royce, 1970

Das Wasserfallmodell ist intuitiv als Abfolge der einzelnen Tätigkeiten im Life-Cycle zu verstehen. Es wird links oben begonnen und arbeitet sich bis nach rechts unten durch. Da man

bei der Überprüfung der einzelnen Phasen auch auf Fehler aus vorhergehenden Phasen stoßen kann, besteht die Möglichkeit auf dem Weg umzukehren und frühere Tätigkeiten wieder aufzunehmen.

Wie dem aufmerksamen Leser vielleicht schon aufgefallen ist, fehlt bis jetzt der Punkt Dokumentation. Dies liegt nicht daran, dass diese unwichtig ist, sondern weil das Dokumentieren zu den Basis-Tätigkeiten gehört und konstant während jeder Tätigkeit zu erfolgen hat. Jede Tätigkeit liefert somit ein Dokument. D.h. eine „Dokumentation im Kopf“ gibt es nicht, sondern nur auf Papier, oder in elektronischer Form. Programmcode zählt mit zur Dokumentation, in ihm müssen auch erklärende Kommentare vorkommen.

4.3.3 Aufwandsverteilung auf die Phasen der Software-Entwicklung

Die Aufwands und Kostenverteilung bei der Softwareentwicklung ist natürlich stark abhängig von dem ausgewählten Modell. Arbeitet man nach dem Trial & Error oder Code & Fix Prinzip, so verbraucht man scheinbar 100% des Aufwands bei der Codierung. Bei einem Vorgehen nach Phasenplan entfallen 40% auf die Phasen vor der Codierung, 20% auf die Codierung und nochmals 40% auf die Phasen nach der Codierung (Zahlen nach Boehm [19]). Bei modernem Vorgehen ist eine Verschiebung des Hauptaufwands in die frühen Projektphasen, der Spezifikation und dem Entwurf festzustellen, dafür sinkt der Testaufwand. Hier hat man eine Verteilung von 60% vor Codierung, 15% für die Codierung und 25% für die Phasen danach.

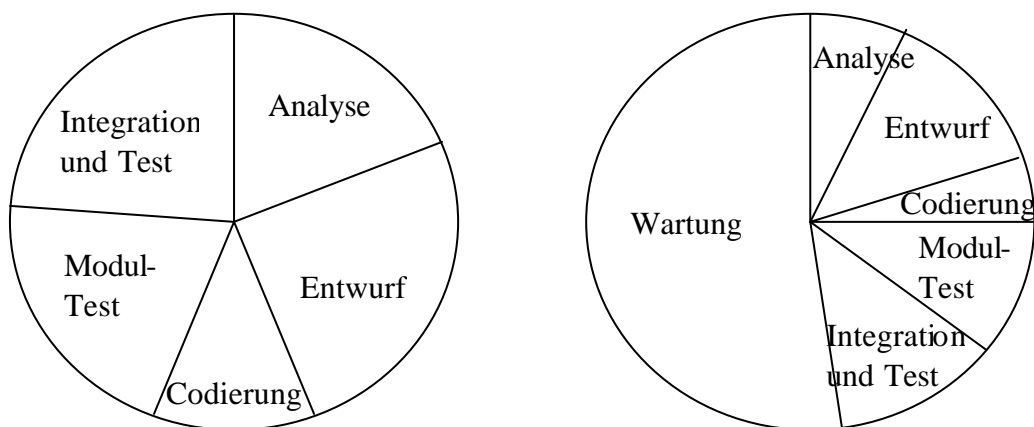


Bild (10) Kostenverteilung – ohne und mit Berücksichtigung der „Wartung“ [19]

Typischerweise werden meist die Kosten, welche die Codierung in Anspruch nimmt, überschätzt. Tatsächlich sind es nur 10-20% der Entwicklungskosten, und umgekehrt werden die Kosten, welche der Test verschlingt, stark unterschätzt. Je weniger nach jedem einzelnen Entwicklungsschritt geprüft wird, desto höher ist der Anteil und er kann auf über 50% steigen.

4.3.4 Zusammenhang zwischen Fehlerentstehung und –entdeckung

Ein Softwareprojekt entsteht bis zur Codierung in einem „top-down“ Verfahren, d.h. von der Analyse über die Spezifikation, den Entwurf und dann zur Codierung. Nach der Codierung haben wir ein „bottom-up“ Verfahren, eine Integration von einzelnen Codeteilen zu Programmen, der Übersetzung, dem Test, usw. bis hin zum Gesamtsystem im Betrieb. Es werden also einzelne Abstraktionsebenen durchschritten. In der Praxis ergab sich folgende Erkenntnis: Fehler werden typischerweise auf derselben Abstraktionsebene entdeckt, auf der sie auch begangen wurden. Beispielsweise zeigt sich ein bei der Analyse begangener Fehler erst im

Betrieb, oder ein Entwurfsfehler wird nicht bei der Codierung sichtbar, sondern erst wenn die einzelnen Module integriert werden sollen.

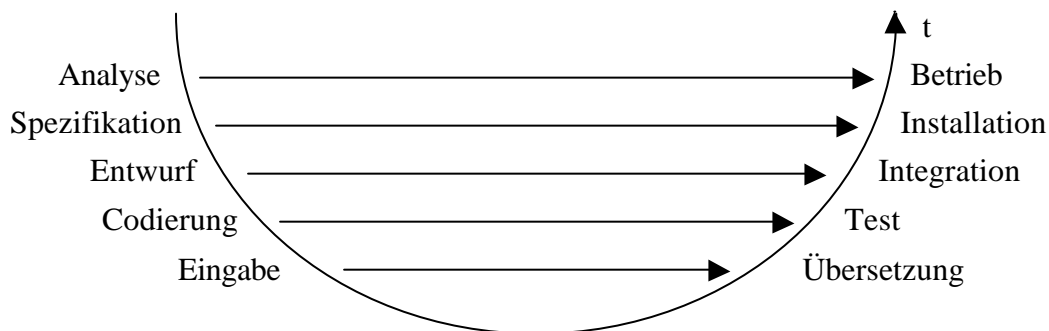


Bild (11) Die Badewannenkurve nach [17]

Die Kosten, die ein Fehler verursacht, sind exponentiell zur Latenzzeit, die vergeht bevor er entdeckt wird, d.h. ein Fehler in den frühen Phasen kann zu astronomisch hohen Kosten führen, wenn er nicht sofort, sondern erst bei der Auslieferung entdeckt wird.

4.3.5 Analyse

Die Analyse dient dazu den Ist-Zustand und den Soll-Zustand am Anfang eines Softwareprojekts festzustellen, bzw. festzulegen. Der Großteil der Informationen die hier zusammengetragen werden, müssen vom Kunden erfragt werden. Damit wird ersichtlich, dass das Resultat der Analyse sehr vom Wissen und Verständnis des Kunden und von der Fähigkeit des Analytikers abhängt, die „richtigen“ Fragen zu stellen. Wegen der starken Abhängigkeit von der Aufgabenstellung und den Vorkenntnissen, lassen sich kaum allgemeine Regeln zur Durchführung der Analyse angeben. Wichtig ist, dass der Kunde das Ergebnis der Analyse nicht nur zur Kenntnis nimmt und abzeichnet, sondern es auch versteht und akzeptiert. Als Ergebnis hat man dann ein Dokument in der Hand, in dem der Ist-Zustand beim Kunden dokumentiert ist, sowie eine Sammlung aller Wünsche und Forderungen des Kunden an das Software-Produkt. Was in der Analyse als Wünsche des Kunden an das Produkt aufgenommen ist, muss nicht unbedingt stimmig und realisierbar sein, sondern stellt nur eine Art Wunschzettel des Kunden dar.

4.3.6 Spezifikation

In der Spezifikation sind dann die Anforderungen an die Software und ihre Schnittstellen definiert. Die Angaben in der Spezifikation haben vollständig, präzise und überprüfbar zu sein. Die hier gestellten Anforderungen sind die Minimalanforderungen an die Software in punkto Funktion und Qualität. Hier sind auch das Verhalten der Software im Sonder- oder Fehlerfall festzulegen. In der Spezifikation wird also festgelegt, was die Software leisten muss, aber nicht wie es realisiert werden soll, d.h. die Spezifikation soll lösungsunabhängig sein. Das so entstehende Pflichtenheft ist die vertragliche Ausgestaltung der Spezifikation. Die Spezifikation ist ein wichtiger Punkt bei der Software-Entwicklung. Sie ist erforderlich für die Abstimmung mit dem Kunden, für den Entwurf, das Benutzerhandbuch, die Testvorbereitung und die Abnahme des Produkts durch den Kunden. Fehlt die Spezifikation, so treten potentiell folgende Probleme auf:

- Anforderungen an die Software bleiben ungeklärt und fallen erst im Betrieb auf.

- Beim Entwurf fehlen entsprechende Vorgaben, und führen so dazu, dass der Entwickler selber Nachforschungen anstellen muss oder aber eigene Annahmen zugrundelegt.
- Der Test kann nicht vorbereitet werden, weil die Korrektheit nicht definiert ist. D.h. ein systematischer Test ist nicht möglich, sondern nur ein „Pseudo-Test“.
- Die Grundlage für die Abnahme und Nachforderungen fehlt. Ein Streit mit dem Kunden ist mangels einer Referenz nicht klar zu entscheiden.

Weil sich aus der Implementierung Funktionen des Programms kaum entschlüsseln lassen, ist eine Wiederverwendung oder Reimplementierung sehr erschwert.

4.3.7 Entwurf

Der Entwurf wird als die zentrale Tätigkeit der Software-Entwicklung betrachtet. Hier wird die Lösungsstruktur der Software festgelegt, und eine Gliederung in handhabbare Einheiten vorgenommen. Die Festlegung der Struktur ist von Bedeutung, weil sie auch noch Jahre nach der Realisierung tragen muss, und somit auch das Wachstum und/oder die Erweiterbarkeit begrenzen oder ermöglichen. Aus diesen Gründen soll der Entwurf:

- Gegen die Spezifikation prüfbar sein.
- Übersichtlich und leicht änderbar sein
- Leicht verwaltbar sein
- Leicht auf die Implementierung abbildbar sein

4.3.8 Codierung

Bei der Codierung wird das, was in den vorhergehenden Phasen erarbeitet wurde, in Programmcode umgesetzt. Hierbei ist darauf zu achten, dass die Grundregeln der Codierung wie „ego-less-programming“, also ein Programmierstil, der nicht den Künstler in seiner Einzigartigkeit hervorhebt, sondern einen Code mit einem Erscheinungsbild, das jeder Leser schnell verstehen kann, eingehalten werden. Des Weiteren sollte es Programmierrichtlinien geben, nach denen codiert wird. Diese sollten u.a. festlegen, wie die Bezeichner zu wählen sind, wie das Layout zu gestalten ist, welche Standardkommentare im Code enthalten sein sollen.

Codierregeln nach Watts S. Humphrey

- Alle Programme sollten mit einem beschreibenden Kopf versehen werden, indem steht zu welchem Projekt das Programm gehört, wer es codiert hat, wann es codiert wurde, und eine kurze Beschreibung was das Programm macht.

- Eine kurze Beschreibung vom Inhalt des Listings.
- Anweisungen zur Nutzung des Programms wie das Deklarationsformat, die Werte und Typen der Übergabeparameter .
- Hinweise zum gültigen Wertebereich der Funktionen, und weitere Bedingungen, die eingehalten werden müssen, um eine korrekte Funktion sicherzustellen.
- Benutzung von aussagekräftigen, beschreibenden Namen für Variablen, Funktionen, Konstanten und anderen Bezeichnern.
- Jeder Parameter muss initialisiert werden, bevor er benutzt wird.
- Alle neuen Pointer müssen vor der Nutzung auf NULL zeigen, und nach Ende der Nutzung wieder freigegeben werden.
- Kommentare sollten den Code so erklären, dass der Leser den Zweck und die Funktion des Codes verstehen kann.
- Vor größeren Programmabschnitten sollten Kommentare sein, die erklären was im folgenden Abschnitt gemacht wird.
- Im Code sollten genügend Freiräume sein, so dass der Code gut lesbar ist.
- Geklammerte Programmabschnitte sollten eingerückt sein, dass klar ersichtlich ist, von wo bis wo ein Block geht.
- Alle Definitionen sollten in Großbuchstaben vorgenommen sein.
- Alle Bezeichner und Schlüsselworte sollten in Kleinbuchstaben geschrieben sein.

Weitergehende Informationen siehe [20].

4.3.9 Test

Definitionen des Tests

- „Testen ist die Ausführung eines Programms mit dem Ziel, Fehler zu entdecken.“
(Myers, 1979)

- „Testen ist die Vorführung eines Programms oder Systems mit dem Ziel zu zeigen, dass es tut, was es tun sollte.“ (Hetzel, 1984)
- „Testen ist die Ausführung eines Programms unter Bedingungen, für die das korrekte Ergebnis bekannt ist, und mit dem des Programms verglichen werden kann; stimmen beide nicht überein, so liegt ein Fehler vor.
Die Entdeckung von Fehlern ist der Zweck des Tests, ein Testfall ist also gut, wenn er hohe Chancen hat, einen Fehler anzuzeigen; tritt dieser Fall ein, so war der Test erfolgreich.
Die Testsituation soll die Einsatzsituation so simulieren, dass aus einem erfolglosen Test auf einen erfolgreichen Einsatz geschlossen werden kann.“ (Ludewig [17])

Die Überprüfung eines Programms mit Hilfe des Tests bringt leider nicht nur Vorteile mit sich, sondern hat auch einige Nachteile. Die Vorteile des Test sind, dass es intuitiv vonstatten geht, es wird alles ausprobiert. Ein Test ist reproduzierbar und damit objektiv, sofern das System deterministisch arbeitet. Er lässt sich, wenn er einmal organisiert ist, sehr billig und oft wiederholen. Die Nachteile des Tests sind, dass er oft überbewertet wird. So ist mittels des Tests keine Korrektheitsaussage zu machen, weil ein vollständiges Testen unmöglich ist. Man denke hier beispielsweise an Rekursion¹ oder verschachtelte Abfragen, wo jede Wertekombination getestet werden müsste. Der Test kann nicht alle Anwendungssituationen erfassen. Beim Test sieht man nur, dass ein Fehler vorhanden sein muss, aber nicht was seine Ursachen sind.

Für jeden Test muss gelten, dass er geplant ist. Zu jedem Testfall gehört ein Soll-Resultat, weil nur ein auffällig falsches Ergebnis ins Auge springt. Es sollten alle spezifizierten Testfälle soweit möglich ausgeführt werden, und nicht sofort beim ersten Fehler mit dem Test aufgehört und mit Debugging begonnen werden.

4.4 Standards und Normen

4.4.1 Einige Datenaustausch-Normen im Bereich Elektrotechnik CAD

Die zunehmende Globalisierung erfordert stärkere Datenkommunikation zwischen den heutzutage oft noch als Insellösungen eingesetzten CAD-Systemen. Wegen der großen Zahl der verfügbaren CAD-Systeme mit unterschiedlicher Datenhaltung, reichen gegenseitige Vereinbarungen nicht aus. Daher wird der Ruf nach einem internationalen Standard lauter [21].

Die Definition von Normen und Standards wurde bisher meist auf nationaler Ebene vollzogen. Der Datenaustausch zwischen verschiedenen CAD-Systemen wurde zumindest hier teilweise realisiert. Neben den Vereinbarungen, die von nationalen Normungsgremien verabschiedet wurden, gibt es auch Standards, die von einzelnen Interessenverbänden oder Gruppen von Industrieunternehmen entwickelt wurden. Anders als die nationalen Normen scheinen sich diese besser zu etablieren.

Es geht bei diesen Standards darum, Produktmodelle in digitaler Form als Ergebnis eines Produktionsschrittes weiterzuleiten, damit diese von einem anderen CAD-System verarbeitet werden können. Hierbei sind in der Regel Post- und Präprozessoren behilflich, wobei der

¹ Rekursion: Definition eines Problems, einer Funktion oder eines Verfahrens durch sich selbst

Postprozessor des sendenden Systems eine Datei in dem jeweiligen Standard erzeugt, die dann zum Beispiel über Netzwerke an den Empfänger gesendet wird. Der Präprozessor des empfangenden Systems konvertiert diese Daten in eine interne Darstellungsform des Zielsystems.

Die Standards definieren, wie die Austauschdatei auszusehen hat und besitzen meistens keine Informationen über den Inhalt der Datei.

Einige Normen sollen nun vorgestellt werden:

4.4.2 IGES

IGES¹ [22] wurde zuerst durch das „National Bureau of Standards“, von „Boeing Corporation“ und von „General Electric Corporation“ entwickelt und durch das „National Bureau of Standards“ 1980 veröffentlicht. Zuerst waren die einzigen Daten, die ausgetauscht werden konnten, grundlegende Elemente wie Punkte, Linien, Bogen und Kreise. Im September 1981 wurde IGES Version 2,0 durch das „American National Standards Institute“ als ANSI Standard-Y14.26M für CAD-/CAM genehmigt. Danach hat man diese Norm ständig erweitert und verbessert. Die neuste Version von IGES ist 5.3 und wurde im März 1995 als NISTIR 5666 [23] veröffentlicht. In der Version 6.0, die sich noch in der Entwicklungsphase befindet, ist eine Erweiterung von IGES vorgesehen, bei der Elemente zum Transfer von B-Rep-Gestaltsmodellen [5] hinzugefügt werden. Es ist geplant, die Entwicklung an IGES einzustellen. Seit 1997 fanden an diesen Standards keine Änderungen mehr statt.

IGES Dateien können entweder binär oder im ASCII-Format sein. In ASCII ist die Datei einfacher zu lesen und zu ändern, aber sie neigt dazu, größer zu sein.

Es gibt zwei Formate

1. ein festes Zeilenlängeformat mit 80 Buchstaben
2. ein freies Format mit komprimierter Zeilenlänge

In den im festen Format gehaltenen Daten gibt es einige Stellen, die in einer in hohem Grade strukturierten Vorgehensweise eingetragen werden müssen. Innerhalb der spezifizierten Spaltenpositionen müssen Daten rechtsbündig angeordnet werden, und sie müssen die Spalten 1 bis 80 besetzen. Im Folgenden kommt ein Auszug einiger Richtlinien, die eingehalten werden müssen, wenn man die IGES Dateien anlegt.

Leerzeichen:

Leerzeichen haben nur Bedeutung in Strings. Ein numerisches Feld, dem ein Leerzeichen zugeordnet wird, besitzt den Default-Wert für dieses Feld. Am Anfang und am Ende von numerischen Konstanten dürfen keine Leerzeichen auftreten. Führende Leerzeichen in den numerischen Konstanten werden ignoriert. Leerzeichen zwischen dem Ende irgendeiner Konstante und der Begrenzung, die der Konstante folgt, sind nicht erlaubt.

¹ IGES: Initial Graphice Exchange Notation

Numerische Konstanten:

Eingebettete Kommata in numerischen Konstanten sind nicht erlaubt. Die absolute Größe einer numerischen Konstante kann den Wert von $2^{(N-1)} - 1$ nicht übersteigen, wobei N die Zahl der Bits ist, die benötigt werden, um eine Realzahl als Integer darzustellen. Reale Konstanten können einfache oder doppelte Genauigkeit besitzen. Gültige Integerzahlen können wie folgt ausgedrückt werden:

1 150 -24567 23457 +23456.

Beispiele für reelle Konstanten:

346.098 0. -.34 0.1E-3 1.E+4.

String Konstanten:

Eine String Konstante ist als willkürliche Reihenfolge der ASCII-Buchstaben definiert. Leerzeichen, Kommata und Zahlen werden als Zeichen innerhalb der Zeichenkette behandelt. String Konstanten werden in der Hollerith-Notation dargestellt. Diese Form besteht aus einer Integerzahl ungleich Null (die Zahl der Zeichen in der Zeichenkette), gefolgt vom Buchstaben H, und der Zeichenfolge.

Gültige Zeichenkettenkonstanten können wie folgt ausgedrückt werden:

3Hr45 12H Hello There 9Htime;.erw

Sequenznummern:

Eine Sequenznummer ist eine Zeichenkette von bis zu sieben Stellen. Sie wird zur Indexierung der Linien innerhalb der verschiedenen Abschnitte der IGES-Datei verwendet. Sequenznummern fangen ab 1, (0000001) für jeden Abschnitt an, und werden sequentiell ohne Unterbrechung zum Ende dieses Abschnitts fortgeführt. Dieses wird hauptsächlich verwendet, um die Anzahl der Linien eines bestimmten Abschnitts anzuzeigen. Sequenznummern müssen in jedem Abschnitt der IGES-Datei verwendet werden. Diese Zahlen werden rechtsbündig in den Spalten 74 bis 80 gehalten. Sequenznummern können führende Nullen oder Leerzeichen enthalten.

Eine IGES-Datei besteht aus sechs Unterabschnitten, die wie folgt erscheinen müssen:

1. Flagabschnitt
2. Startabschnitt
3. Globaler Abschnitt
4. Verzeichniseintragabschnitt
5. Parameterdatenabschnitt

6. Endabschnitt

Die Hauptfunktionalität von IGES besteht eigentlich darin, Graphiken zu übertragen. Damit lassen sich technische Zeichnungen (etwa Flächen- und Kantenmodelle) beschreiben.

Ab der Version 5.1 hat man die Zielsetzung verallgemeinert. Zum einen hat man beschlossen, IGES zur Bereitstellung von Informationsstrukturen, die zur digitalen Repräsentation und zur Kommunikation von produktdefinierenden Daten verwendet werden können, zu erweitern, und zum anderen sollte der Austausch zwischen verschiedenen CAD-Systemen möglich sein.

Obwohl sich einige Erweiterungen von IGES in Richtung Elektrotechnik entwickelt hatten, ist dieser Standard weitgehend für den Maschinenbau bestimmt. Deshalb hat IGES in der elektrotechnischen Branche kaum Bedeutung.

4.4.3 SET

SET¹ [24] ist ein französischer Standard zum Austauschen und Archivieren von CAD-Daten. Er wurde als neutrales Format zum Austausch von Daten zwischen unterschiedlichen CAD-Systemen bei der Firma Aérospatiale 1983 entwickelt. Er sollte eine Alternative zu IGES bilden. Das Ziel war, eine zuverlässigere Alternative zu IGES zu entwickeln. 1985 wurde SET ein offizieller französischer Standard. SET wurde 1989 überprüft und verbessert. Der Grundgedanke hinter dem Standard war, dass er alle anfallenden Informationen, die auftreten können, darstellen sollte. Von weiterer Wichtigkeit war es, ein eindeutig definiertes und kompaktes Format zu haben, das auch in der Zukunft der CAD-/CAM Industrie genüge tut.

Die Struktur von SET basiert auf einer hierarchischen Struktur mit drei Levels. Allgemeine Informationen der Blöcke oder Zusammenstellungen werden in einem so genannten Wörterbuch gespeichert.

Zusammenstellungen:

Eine SET Datenzusammenstellung beschreibt einen bestimmten Teil von Information, wie zum Beispiel ein mechanisches Bauteil. Eine SET-Datei kann eine oder mehrere Datenzusammenstellungen enthalten.

Block:

In der Definition der SET-Dateien, werden die Blöcke durch „@“ gekennzeichnet, gefolgt von der Nummer des Blockes, der seinen Typ definiert. Auf diese Nummer folgt die Referenznummer des Blockes. (d.h. in einer Datei mit n Blöcken, gibt es Sequenznummer zwischen 0 und n). Nach diesem Eintrag folgen die Sub-Block- und die Wörterbucheinträge, die im Block benutzt werden. Ein SET-Datenblock ist eine Basis-Entity, die aus Definitions- oder Steuerdaten besteht. Solche Entities können geometrische Objekte, wie Punkte, Linien oder andere Entities, wie Matrizen, Zeichnungen sein.

Sub-Block:

Ein Sub-Block besteht aus einem Bezeichner und einer Liste von Daten, die zur Beschreibung der Entity gehört, die durch den Datenblock definiert wird. Die unterschiedlichen Parameter

¹ SET: Standard d'Exchange et de Transfer

innerhalb des **Sub-Blocks**, wie beispielsweise Koordinaten, werden durch ihre Werte dargestellt. Ein **Sub-Block** beginnt mit dem Zeichen „#“, gefolgt von seiner Typennummer, eventuellen Verweisen zum Wörterbuch, und von Parametern, die in diesem **Sub-Block** anwendbar sind.

Wörterbuch:

Das Wörterbuch ist ein Satz vorbestimmter Parameter, die in der Spezifikation des Standards festgelegt sind. Der Zugriff auf sie erfolgt durch ein „:“, gefolgt von der Referenznummer im Wörterbuch.

Verweise unter Entities in einer SET-Datei werden durch Zeiger realisiert. Sie können entweder direkt von den Blöcken, über Sub-Blöcke oder über Wörterbucheinträge verwendet werden. Zeiger zu anderen Blöcken werden durch ein „!“ gekennzeichnet, auf das die Sequenznummer der Blöcke folgt.

Aufgrund der verbalen Art und Weise der Informationselemente, die nicht allzu sehr formalisiert sind, kann eine automatische Prüfung auf Konsistenz nicht erfolgen. Ähnlich zu IGES fehlen Definitionen der Untermengen, so dass auch hier die gleichen Probleme auftreten. Die Entwicklung an SET wird wohl mit der Durchsetzung von STEP auf dem Markt eingestellt werden.

4.4.4 VNS

Die Entwicklung der verfahrensneutralen Schnittstelle für Schaltplandaten (VNS) wurde im Jahre 1989 aus dem Bedürfnis der Energieversorgungsunternehmen und des Verbandes der deutschen Automobilindustrie nach dem beleglosen Austausch von Schaltungsunterlagen der elektrotechnischen Anlagendokumentation begonnen. Die erste Stufe wurde in kurzer Zeit von einem Arbeitskreis entwickelt, der aus Anbietern und Anwendern von ECAD-Systemen bestand. Diese Version unterstützte den Austausch von Stromlaufsymbolen und Plänen [25]. 1991 wurde dieser Vorschlag als deutsche Vornorm DIN 40950 [26] veröffentlicht.

Der wesentliche Nachteil an diesem Verfahren war, dass die Beschreibung der Produktdaten im Vordergrund stand, nicht die Produktdaten selbst.

Die Daten werden in VNS aus der Graphik abgeleitet, leider auch mit allen nachteiligen Folgen, die diesen Ansatz begleiten.

Außerdem wird bei VNS lediglich die Syntax genormt, nicht die Semantik, da VNS eine reine Formatspezifikation ist.

4.4.5 EDIF

Der Mangel eines allgemeinen Austauschformates in der Elektronik war der Anstoß für die Entwicklung von EDIF¹ [27] durch die Firmen Motorola, Texas Instruments, National Semiconductor, Mentor, Daisy und Tektronix. Diese haben 1983 das Projekt initiiert, wobei sie sich auf den Entwurf und die Herstellung von integrierten Schaltkreisen spezialisierten. Der erste Standard wurde 1985 veröffentlicht. Die aktuelle Version ist 4.0.0. und ist als ANSI Norm [28] veröffentlicht (ANSI/EIA 682-1996). Sie erweitert die in 1993 veröffentlichte Version 3.0.0 hauptsächlich um die Unterstützung von PCB² und PCs.

¹ EDIF: Electronic Design Interchange Format

² PCB: Printed Circuit Board

Bei der Entwicklung stand die Einhaltung der Unabhängigkeit, sowohl zwischen Rechnern und Betriebssystemen, als auch zwischen verschiedenen CAD-Systemen im Vordergrund. Es sollte leicht verständlich und einfach handhabbar sein. Bei der Entwicklung bedachte man die Erweiterbarkeit und Aufwärtskompatibilität. EDIF sollte sich in der Elektro-Industrie als der wichtigste Standard durchsetzen.

In EDIF ist es möglich, verschiedene Ebenen auf Schaltungsbeschreibungen zu definieren:

- Verhaltensorientierte Beschreibungen
- Graphisch dargestellte Schaltschemata
- Netzlisten
- Layout-Beschreibungen

Die Dokumentation von Schaltungen besteht aus ACSII Zeichen, so dass sie auch von Menschen lesbar ist. Sie kann aufgrund ihres Umfangs allerdings nur maschinell erzeugt bzw. weiterverarbeitet werden.

Der Vorteil von EDIF gegenüber anderen Standards besteht darin, dass EDIF zum Teil die Phasen des Produktionszyklus unterstützt und für noch fehlende, später hinzukommende Bereiche, erweiterbar ist. Die produktdefinierenden und –beschreibenden Daten werden bei EDIF in den Vordergrund gestellt und die graphische Präsentation von Produktdaten daraus abgeleitet.

Ein Problem, das auch EDIF nicht löst, ist die Beschränkung auf die speziellen Bedürfnisse der ECAD-Systeme. Außerdem ist EDIF, wie alle vorgestellten Standards, eine Formatspezifikation, bei der die mangelnde Formalisierung der Informationselemente zum Teil verschiedene Darstellungs- und Interpretationsmöglichkeiten erlaubt.

4.4.6 Internationaler Standard für Produktdatenaustausch STEP

Durch die Globalisierung und steigende Komplexität der Programme gewinnt der Austausch von Produktdaten immer mehr an Bedeutung. Die Vielzahl von ECAD-Systemen, die sich heutzutage auf dem Markt befinden, mit den unterschiedlichsten Exportformaten erschwert den Datenaustausch. Die spezifischen Informationen eines ECAD-Systems müssen beim Austausch der Daten transparent bleiben. In der Vergangenheit wurden bereits einige solcher Austauschformate (siehe oben) vorgestellt. Neben diesen genormten Datenformaten existieren Quasistandards, wie beispielsweise das DXF-Format von AutoCAD, das sich auch in anderen ECAD-Systemen etabliert hat.

Dabei muss jedes System, das dieses Format unterstützt, einen Präprozessor zum Schreiben, sowie einen Postprozessor zum Lesen der Datei bereitstellen.

Ein anderer Ansatz besteht darin, direkte Übersetzer in die ECAD-Systeme zu integrieren. Dies bedeutet jedoch, dass man in einer Umgebung mit n Systemen $n(n-1)$ Übersetzer benötigt. Wird jedoch ein neutrales Datenaustauschformat benutzt, benötigt man lediglich $2n$ Übersetzer.

Um die oben genannten Probleme der bestehenden Austauschformate in den Griff zu bekommen, hat man sich bereits 1984 dafür entschieden, ein Projekt zu initiieren, welches das Ziel hatte, ein neues Produktdatenaustauschformat zu definieren. Dieses Projekt hatte den Namen STEP¹ ISO 10303 [29], [30], [31] bekommen. Die damals an STEP gestellten Anforderungen waren:

- STEP muss ein Produkt vollständig beschreiben können.
- Unterstützung des gesamten Produktentwicklungsprozesses muss gewährleistet sein.
- Durch Auswahl einer geeigneten Beschreibungssprache (z.B. EXPRESS) muss eine Produktmodellierung möglich sein.
- Kompatibilität von STEP zu anderen Standards
- Plattformunabhängigkeit
- Durch Anwendungsprotokolle soll die Semantik für verschiedene Technologiebereiche eindeutig definiert werden.

STEP wurde von Anfang an so entwickelt, dass jederzeit Erweiterungen oder Verbesserungen daran vorgenommen werden könnten. Ein wichtiger Punkt bei der Spezifizierung war der Gebrauch der objektorientierten Methoden. Eine Beschreibungssprache, mit der sich STEP anwenden lässt ist EXPRESS. EXPRESS-G realisiert die graphische Darstellung der Module.

EXPRESS besteht aus folgenden Konstrukten:

- **Schemata:**
Definition von einem EXPRESS-Modul und sinnvolle Unterteilung der Daten.
Man kann ein Schema allgemein definieren, dann in ein anderes Schema wechseln und dort weiter arbeiten.
- **Typen:**
Es existieren vordefinierte Typen wie NUMBER, INTEGER, REAL, STRING, LOGICAL, BOOLEAN und BINARY. Außerdem können Typen frei definiert werden. Weiterhin gibt es die Möglichkeit über Anweisungen SUBTYPE bzw. SUPERTYPE Eigenschaften von Entities zu vererben.
- **Konstanten:**
Konstanten sind deklarierte Größen.

¹ STEP: Standard for The Exchange of Product model Data

- **Funktionen:**
Unterprogramme, die einen Wert zurückliefern müssen
- **Prozeduren:**
Unterprogramme, die im Gegensatz zu Funktionen, keinen Rückgabewert zurückliefern müssen
- **Regeln:**
Konventionen, die hinzugezogen werden, um Sachverhalte aus der realen Welt zu modellieren

Die systemübergreifende Funktionalität wird in STEP dadurch erreicht, dass man hier Anwendungsprotokolle definiert. Durch ein neutrales Datenmodell einer bestimmten Branche, beispielsweise AP212 (ISO-Norm 10303-212) [29] für die Elektroindustrie oder AP214 (ISO-Norm 10303-214) [29] für die Automobilindustrie, können Daten zwischen verschiedenen Unternehmen respektive deren Systemen ausgetauscht werden. Damit ist ein Weg zum Informationsaustausch zwischen verschiedenen CAD-Systemen über System- und Unternehmensgrenzen hinweg geebnet.

Ein Anwendungsprotokoll besteht im Wesentlichen aus den folgenden Teilen :

- **Scope:**
Eine kurze Beschreibung der Datenquelle
- **Requirements:**
Definition der Informationseinheiten
- **Application Interpreted Model:**
Schema, das durch Interpretation der integrierten Ressourcen und Zuordnung je nach Anforderungen des APs erstellt wird
- **Conformance Requirements:**
Methoden zum Testen der Implementierungen, Richtlinien zur Erstellung von „abstract test suites [29]“

4.4.7 Anwendungsprotokoll 212 (AP 212)

Zum Beschreiben der elektronischen Erzeugnisse wurde das Anwendungsprotokoll AP212 (Electrotechnical Design and Installation) entwickelt. Es besteht aus acht Teilen, mit deren Hilfe eine umfassende Beschreibung der Datenmodelle und Methoden gewährleistet wird:

1. Allgemeine Beschreibung des Produkts und seiner Struktur

- Informationen über die Beschaffenheit des Produkts.

- Produkte, die keinen Strom erzeugen, verwenden oder umwandeln, werden als nicht elektrisch eingestuft.
- Elektrotechnische Produkte können aus elektrischen und nichtelektrischen Bauteilen bestehen.

2. Verbindungen und Netze

- Elektronische Produkte können sowohl elektrische als auch nichtelektrische Knoten besitzen.
- Kommunikation und Übertragung zu anderen Produkten findet nur über diese Knoten statt.
- Die Ein- und Ausgänge von Produkten werden im Interface zusammengefasst.
- Die Verbindung zu den Knoten wird wiederum durch Produkte hergestellt. Dadurch können sehr tiefe Verschachtelungen entstehen.

3. Funktionsbeschreibung

- Funktionen von Teilen können beschreiben werden.
- Erlaubt die Angabe von Voraussetzungen für eine spätere Verbindung mit anderen Produkten.
- Unabhängigkeit der Beschreibung von der verwendeten Technologie.

4. Planung der Verkabelung und Installation

- Physikalische Position eines Produktes innerhalb einer Konstruktionseinheit.
- Verbindungswege innerhalb einer Konstruktionseinheit.
- Produkte, die zur Einrichtung von Verbindungen benötigt werden.
- Teile, die zu Installation von anderen Produkten nötig sind.

- Basiert auf einem 3D-Modell, in dem Verbindungswege durch Knoten, Abschnitte und Kurven definiert sind.

5. Dokumentation und grafische Darstellung

- Basiert auf einem 2D-Modell zur Darstellung der Schaltschemata und Symbole und ist damit Untermenge des ISO 10303-201-Protokolls.

6. Zuordnung von Datenelementen zu Objekten

- Definiert Möglichkeiten zum Anbinden von Datenelementen aller Art zu unterschiedlichen Objekten.
- Jedes Element wird beschrieben durch Name, Wert, Datentyp und Definition sowie der Referenz zur Quelle.

7. Referenzen zu externen Dateien

- Stellt die Möglichkeit zur Verfügung, STEP unbekannte Dateiformate zu definieren. Dadurch kann man diese Daten in die Konstruktion übernehmen.

8. Versionsbehandlung und Konfiguration

- Möglichkeiten zur Angabe von Versions- und Indexnummern, Prüfstatus und ähnlichem.
- Die Voraussetzung für einen Einsatz des STEP-Datenmodells AP212 ist eine Untersuchung der Firmenstruktur und Firmendaten, bei der geprüft wird, ob sich die Bedürfnisse und Projektabläufe des entsprechenden Unternehmens auf die AP212-Datenobjekte abbilden lassen.

5 Auswahl der Entwicklungsumgebung

Aufgrund unserer Festlegung auf einen möglichst systemunabhängigen Entwicklungsansatz, wird im Folgenden, abgesehen von den Systemen, nur auf Programmiersprachen und Werkzeuge eingegangen, die diesem Anspruch gerecht werden. D.h. es werden beispielsweise nur Programmiersprachen betrachtet die eine gute Portabilität haben, und dadurch keine Festlegung auf eine Systemumgebung erfolgt.

5.1 Systeme

5.1.1 Betriebssysteme

Prinzipiell eignen sich alle auf dem Markt verfügbaren Betriebssysteme, die auch von ECAD-Systemen unterstützt werden. Bei der Auswahl des Produkts muss vor allem auf die Verfügbarkeit geachtet werden. Auf Grund dessen bietet sich an, die Entscheidung zwischen einem UNIX Derivat und einer aktuellen Version von Windows [10] zu fällen. Im Folgenden werden die zwei Betriebssystemfamilien näher beschreiben.

Windows Familie [10]

Windows 95[®]

Im August 1995 erschien Windows 95[®]. DOS muss bei diesem Windows nun nicht mehr vorher installiert werden, obwohl es aus Kompatibilitätsgründen immer noch als Unterbau des ganzen Systems fungiert. TCP/IP¹ ist jetzt im Standardumfang enthalten, wodurch Rechner mit Windows 95[®] als Clients für verschiedenste Netzwerke durchaus brauchbar werden. Die alte 8+3 Namenskonvention gehört der Geschichte an, für leichten Zugang ins Internet ist auch gesorgt.

Mit dem Service Release 2 werden USB²-Unterstützung sowie mit FAT32³ ein erweitertes Dateisystem eingeführt. Die Benutzeroberfläche wurde im Vergleich zu seinen Vorgängern komplett geändert, Windows 95[®] erweckt beim Anwender den Eindruck, als sei es bereits weitgehend objektorientiert.

Windows NT 4.0[®]

Im dritten Quartal 1996 erscheint auch NT 4.0 (als Workstation und Server-Version) mit der neuen Benutzungsoberfläche von Windows 95[®]. Um die Bildschirmausgabe zu beschleunigen, wurde der entsprechende Treiber in den Kernel gelegt. Dies führte zu Schwierigkeiten bei einigen Grafikkarten. Zudem wurden Änderungen im RAS⁴ durchgeführt.

¹ TCP/IP: Transmission Control Protocol / Internet Protocol

² USB: Universal Serial Bus

³ FAT: File Allocation Table, FAT32 erhöht im Vergleich zu FAT die Anzahl der Adressbits. Mit FAT32 kann auf bis zu 2 TB (Terrabytes) große Festplatten zugegriffen werden

⁴ RAS: Remote Access Service

Bis heute wurden 6 Service-Packs nachgeschoben, wodurch diese Version als hochstabil bezeichnet werden kann. Die Auslieferung wurde, obwohl der Nachfolger Windows 2000[®] bereits auf dem Markt war, erst im September 2001 eingestellt.

Windows 98[®]

Windows 98[®] wird im Juni 1998 veröffentlicht. Der Internet Explorer 4.0[®] wird (angeblich) vollkommen in die grafische Benutzeroberfläche integriert, was Microsoft so manchen Ärger mit dem Browser-Marktführer Netscape einbringt. Aber durch diese enge Integration in Windows können Internet-Seiten auch auf dem (Active-) Desktop platziert werden, wodurch sich deren Inhalte im Falle einer Online Verbindung automatisch aktualisieren. Für das neue FAT32 Dateisystem (unterstützt Festplatten größer 2 GB), ist ein Hilfsprogramm vorhanden, das die Konvertierung bestehender Installationen von FAT16 nach FAT32 erlaubt. Zudem wurde das Powermanagement (ACPI¹) überarbeitet, das vor allem die Verwendung von Windows mit Notebooks vereinfacht.

Neue Hardware wie DVD², Firewire und AGP³ wird unterstützt. Erwähnenswert ist auch, dass das Treiber Modell erstmals mit dem High-End Betriebssystem Windows NT kompatibel gemacht wurde.

Windows 98 SE[®] (Zweite Ausgabe)

Die Auslieferung von Windows 98[®] SE erfolgt im Mai 1999. In dieser Version ist bereits der neue Internet Explorer 4.0[®] integriert. Während für die Vorgängerversionen Windows 95[®] und Windows 98[®] die sogenannten Plus-Pakete separat zugekauft werden mussten, sind diese in SE bereits integriert. Verschiedene Programme wurden überarbeitet (Paintbrush, Defragmentierer)

Windows 2000[®]

Der lange als Windows NT 5.0[®] angekündigte Nachfolger von Windows NT 4.0[®] wurde im Jahr 2000 ausgeliefert. Fast vollständig in C++ kodiert, mit 29 Millionen Zeilen (!) eines der größten Softwareprodukte, die bis dahin jemals produziert wurden. Die Boot Sequence wurde überarbeitet, die Speicherverwaltung geändert, die Netzwerkfähigkeiten verbessert und endlich auch die USB-Unterstützung implementiert.

Das von Windows 98[®] her bekannte Filesystem FAT32 wird jetzt auch von Windows 2000[®] unterstützt. Der neue "Windows Installer" soll die saubere Deinstallation von Programmen gewährleisten und die Nachinstallation von benötigten Modulen vereinfachen.

Windows ME[®]

Das letzte Windows (ME steht für Millennium Edition), das MS/PC-DOS⁴ als Grundlage benutzt, wird ausgeliefert. Der DOS-Modus selbst wurde bei ME allerdings entfernt. Mit ME endet zudem der 16Bit-basierte Zweig des Windows Baums. Als wichtigste Neuerung in ME sei das "System Recovery" angeführt. Diese Funktion stabilisiert das Betriebssystem, indem es sicherstellt, dass die Systemdateien den für einen ordnungsgemäßen Betrieb benötigten

¹ ACPI: Advanced Configuration and Power Interface

² DVD: Digital Versatile Disc

³ AGP: Accelerated Graphics Port

⁴ DOS: Disc Operating System

Versionen entsprechen. Ansonsten wurden lediglich einige Multimedia Erweiterungen (u.a. Videoschnitt) als Verbesserungen gegenüber der Vorgängerversion Windows 98[®] SE eingebaut.

Unix

Unix soll dem Anwender helfen, die anfallenden Aufgaben, so genannte Prozesse, besser koordinieren zu können. Dazu gehört unter anderem, Dienste zur Verfügung zu stellen oder das Ausführen von Verwaltungsaufgaben, wie die Speicherung von Daten. Dabei stehen 3 wesentliche Komponenten zur Verfügung, die im weiteren Abschnitt etwas näher veranschaulicht werden sollen: der Kernel, die Shell und die Utilities.

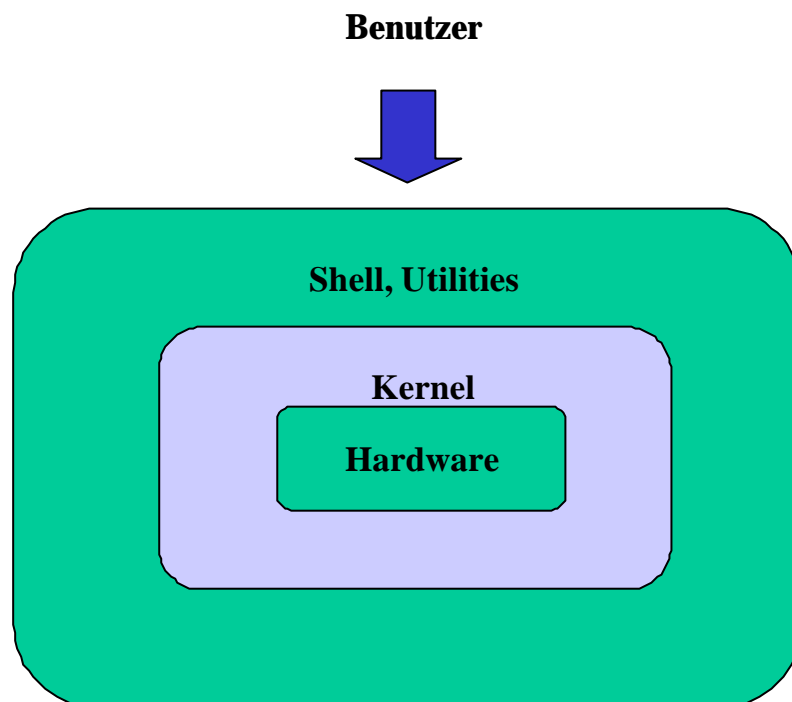


Bild (12) Das Schalenkonzept von UNIX

Der Kernel ist der Teil des Betriebssystems, der mit der Hardware kommuniziert. Er soll grundlegende Dienste wie das Öffnen, Schließen, Lesen oder Schreiben einer Datei ermöglichen.

Darüber hinaus gehört zum Aufgabenbereich die Speicher- bzw. Prozessverwaltung. Programme können die Dienste des Kerns durch Systemaufrufe nutzen, die im Betriebssystem in Form von Bibliotheken integriert sind.

Die Shell ist ein wesentlicher Bestandteil von UNIX, und ist die Schnittstelle zwischen dem Innenleben von UNIX und dem Benutzer. Sie fungiert als Schale um den UNIX-Systemkern und ist die Basis für die Mensch-Maschine-Kommunikation und als Kommandointerpreter, das heißt konkret, von ihr werden die Anwenderprogramme, Benutzereingaben und die UNIX-Utilities abgehandelt. Die Shell „übersetzt“ Kommandos in Aufrufe von Systemfunktionen an den UNIX-Kernel und dieser führt die entsprechenden Anweisungen aus und liefert als Ergebnis Statusmeldungen zurück, die von der Shell ausgewertet werden. Sie dient auch

als Interpreter, der die Programme, die in der Shell-Sprache geschrieben sind, abarbeiten kann. Dadurch können auch leichte Programmieraufgaben, unter Ausnutzung der Fähigkeiten der im nächsten Abschnitt verdeutlichten UNIX-Utilities, erledigt werden.

UNIX enthält auch über einhundert Dienstprogramme, die so genannten Utilities, die einzeln oder auch in Kombination miteinander ausgeführt werden können. Die Utilities sind eine Art Werkzeugkasten, dessen Werkzeuge sinnvoll einzeln genutzt oder aber auch kombiniert werden können. Dem System können auf einfache Weise weitere Anwenderprogramme hinzugefügt werden, die auf den bestehenden Utilities aufbauen und deren Funktionen verwenden können. Einige Beispiele für Utilityprogramme sind Kommandos zum Anzeigen des Inhalts von Dateien, das Sortieren von Daten oder zum Suchen von bestimmten Dateien in Verzeichnissen, welche die gewünschten Kriterien wie (Größe, Datum, letzter Zugriff usw.) erfüllen.

Um an Informationen über die Vielzahl der verfügbaren Kommandos zu gelangen, können die „Manualeiten“, quasi ein integriertes Handbuch im System, per Kommandoeingabe *man* aufgerufen werden. Hier werden Kurzbeschreibungen und eine Anleitung aufgelistet, die ebenfalls zur Lösung von Problemen/Fragen beitragen können. Kommandos werden unter UNIX entweder im Hinter- oder im Vordergrund gestartet.

Ein sehr nützliches Hilfsprogramm ist *crontab*, mit dem man Prozesse, wie das Ausführen/Starten eines oder mehrerer Kommandos, zu gewünschten Zeiten durchführen lassen kann. In der *crontab*, einer Tabelle, in der man bequem zeilenweise Aufgaben bzw. Aufträge editieren kann, werden dann mittels des Prozesses *cron*, die anstehenden Kommandos ausgeführt. Es gibt auch ein bestimmtes Format, in dem die Einträge vorliegen müssen. Die wichtigsten Kriterien hierbei sind Minuten, Stunden, Tag, Monat und auch ein ganz bestimmter Wochentag (Montag bis Sonntag).

Im Moment existieren verschiedene Unix Derivate auf dem Markt:

- SunOS bzw. Solaris 1.x der Firma Sun Microsystems [32]
- Solaris 2.x der Firma Sun Microsystems
- IRIX der Firma Silicon Graphics Inc. [33]
- HP-UX ab der Version 9.0.7 der Firma Hewlett-Packard [34]
- AIX ab der Version 3.2.5 der Firma IBM [35]

5.1.2 ECAD-Systeme

Auf dem heutigen Markt existiert eine Vielzahl an ECAD-Systemen, die eine Integration des Variantenmoduls, das in der prototypischen Realisierung implementiert werden soll, ermöglichen könnten. Die ECAD-Systeme der verschiedenen Anbieter sollen hier nicht weiter aufgeführt werden.

Voraussetzung für das Forschungsprojekt ist allerdings, dass ein solches ECAD-System über eine API verfügt. Es muss eine Makrosprache zur Verfügung stellen, mit der man es über das API steuern könnte. Die Kommandosprachen bieten dem Kunden Kontrollfunktionen und Steuerungsmöglichkeiten für die Interaktion mit dem System. Meistens existieren diese Programme in Form von Kommandointerpretern.

Nur so kann man das Variantenmodul an das ECAD-System koppeln, ohne in den Kern des ECAD-Systems eingreifen zu müssen, da die internen Schnittstellen nicht bekannt sind.

5.2 Programmiersprachen und Werkzeuge

5.2.1 C++

Wie entstand C++

Als sich die objektorientierte Methodik bei Analyse, Entwurf und Programmierung durchzusetzen begann, nahm Bjarne Stroustrup [36] die populärste Sprache für die kommerzielle Software-Entwicklung, nämlich C, und erweiterte sie um Merkmale für die objektorientierte Programmierung. Er schuf C++, das zunächst einmal für eine Handvoll Entwickler bei AT&T entwickelt worden war, aber in weniger als einem Jahrzehnt zu einer der populärsten Programmiersprachen für schätzungsweise mehr als eine Million Entwickler auf der ganzen Welt gereift ist.

Objekte

C++ ist nicht nur das bessere C, sondern bietet dem Entwickler die Möglichkeit, objektorientiert zu programmieren. Dies verlangt neben dem Erlernen neuer Sprachelemente auch eine neue "objektorientierte" Denkweise.

Herkömmliche Softwareentwicklung bestand oftmals darin, zur Lösung eines vorgegebenen Problems Algorithmen zu entwerfen und diese in Prozeduren zu gießen, die in einer Programmiersprache - wie zum Beispiel C - formuliert sind. Man spricht daher auch von prozeduraler Programmierung. Betrachtet man jedoch die reale Welt, so stellt man fest, dass die Dinge sich hier nicht in einer abstrakten prozeduralen Weise bewegen. Diesen Bruch zwischen realer Welt und Softwareentwicklung versucht der objektorientierte Ansatz zu überwinden. Analysiert man seine materielle Umgebung, so stellt man fest, dass diese im Wesentlichen aus Objekten besteht, die in verschiedener Art und Weise miteinander agieren.

Wenn man die diese Erkenntnis aus der realen in die Software-Welt transformiert, so kann man formulieren:

Ein Software-Objekt ist ein Bündel aus Attributen und darauf bezogenen Methoden.

Klassen

Wenn wir verschiedene Objekte der realen Welt genauer betrachten, so stellen wir fest, dass gleichartige Objekte ähnlich sind: Alle besitzen irgendwelche gemeinsamen Attribute, beispielsweise Farbe. Es muss also einen Bauplan geben, der beschreibt, wie ein Objekt grund-

sätzlich auszusehen hat. Alle Instanzen sind nach diesem Bauplan erstellt worden - deswegen ist es uns auch möglich, ein beliebiges Objekt zu benutzen, wenn wir die Kenntnis über die Klasse besitzen. Übertragen auf die Software-Welt bedeutet dies:

Eine Klasse ist ein Bauplan, welcher die Attribute und Methoden definiert, die alle Objekte einer bestimmten Art besitzen.

Vererbung

Vererbung erlaubt die Definition neuer Klassen auf der Basis von bestehenden Klassen. Dies ist ein grundlegendes Konzept objektorientierter Designs. Der Begriff der Klasse wurde als eine Art Bauplan für Objekte erklärt. Es fällt auf, dass es hier verschiedene Arten von gleichartigen Objekten gibt. Sie haben alle gewisse gemeinsame Eigenschaften. Zusätzlich zu diesen Gemeinsamkeiten bringen sie aber auch neue Eigenschaften ein.

In objektorientierter Sprache könnte man also sagen: Die Sub-Klassen erben von der Super-Klasse gemeinsame Eigenschaften und fügen zusätzliche hinzu. Allgemein gilt:

- Klassen können definiert werden in Abhängigkeit von anderen Klassen: "A ist eine Art von B". In diesem Fall ist B die Basisklasse von A.
- Eine Klasse kann auch von mehreren Klassen erben: A ist eine Art von B und C (Mehrfachvererbung).
- Jede Klasse erbt die (öffentlichen) Attribute und Methoden ihrer Basisklasse(n).
- Jedoch kann jede Klasse eigene Variablen und Methoden hinzufügen.

Es ist wichtig zu verstehen, dass Vererbung nur in eine Richtung läuft: Eine Instanz der Sub-Klasse A erbt die Attribute seiner Basisklasse B. B hingegen kann die (zusätzlichen) Attribute von A nicht erben. Wenn in C++ ein Objekt a der Klasse A definiert wird und die Klasse A von B abgeleitet ist, dann kann a jederzeit per Cast in ein Objekt vom Typ B umgewandelt werden. Die Umkehrung gilt nicht: Ein Objekt b der Klasse B kann in diesem Fall nicht in den Typ A umgewandelt werden!

5.2.2 Java

Java wurde von der Firma Sun [32] entwickelt und erstmals am 23. Mai 1995 als neue, objektorientierte, einfache und plattformunabhängige Programmiersprache vorgestellt. Sun besitzt das Schutzrecht auf den Namen Java und stellt damit sicher, dass nur "100 % richtiges Java" diesen Namen tragen darf. Die Sprache ist aber für alle Computersysteme verfügbar. Java geht auf die Sprache Oak zurück, die 1991 von Bill Joy, James Gosling und Mike Sheridan im Green-Projekt entwickelt wurde, mit dem Ziel, eine einfache und plattformunabhängige Programmiersprache zu schaffen, mit der nicht nur normale Computer wie Unix-Workstations, PCs und Apple programmiert werden können, sondern auch die in Haushalts- oder Industriegeräten eingebauten Micro-Computer, wie z.B. in Waschmaschinen und Videorekordern, Autos und Verkehrsampeln, Kreditkarten und Sicherheitssystemen und vor allem auch in TV-Settop-Boxes für "intelligente" Fernsehapparate.

Allgemein anerkannt wurde Java aber erst seit 1996 in Verbindung mit Web-Browsern und Internet-Anwendungen sowie mit der Idee eines NC¹, der im Gegensatz zum PC nicht lokal installierte, maschinenspezifische Software-Programme benötigt, sondern die Software in Form von Java-Klassen dynamisch über das Netz (Intranet) von einem zentralen Server laden kann. Diese Idee wurde später zu einem allgemeinen ASP² erweitert.

Der Name wurde nicht direkt von der indonesischen Insel Java übernommen sondern von einer bei amerikanischen Programmierern populären Bezeichnung für Kaffee.

Die wichtigsten Eigenschaften von Java sind:

- Plattformunabhängigkeit
- Objektorientiertheit
- Syntax ähnlich wie bei C und C++
- umfangreiche Klassenbibliothek
- Sicherheit von Internet-Anwendungen

Bei Java-Programmen muss zwischen zwei grundsätzlichen Arten unterschieden werden: Applikationen und Applets

Applikationen

Java-Applikationen sind Computer-Programme mit dem vollen Funktionsumfang, wie er auch bei anderen Programmiersprachen gegeben ist. Applikationen können als lokale Programme auf dem Rechner des Benutzers laufen oder als Client-Server-Systeme über das Internet bzw. über ein Intranet oder als Server-Programme (Servlets, CGI-Programme) auf einem Web-Server.

Technisch gesehen zeichnen sich Java-Applikationen dadurch aus, dass sie eine statische Methode main enthalten.

Applets

Java-Applets werden innerhalb einer Web-Page dargestellt und unter der Kontrolle eines Web-Browsers ausgeführt. Sie werden meist über das Internet von einem Server geladen, und spezielle Sicherungen innerhalb des Web-Browsers ("Sandkasten", sandbox) sorgen dafür, dass sie keine unerwünschten Wirkungen auf den Client-Rechner haben können. So können Applets z.B. im Allgemeinen nicht auf lokale Files, Systemkomponenten oder Programme zugreifen und auch nicht auf Internet-Verbindungen außer zu dem einen Server, von dem sie geladen wurden.

¹ NC: Network Computer

² ASP: Application Service Providing

Technisch gesehen zeichnen sich Java-Applets dadurch aus, dass sie Unterklassen der Klasse Applet sind.

JavaScript ist nicht Java

JavaScript ist eine Skript-Sprache, die in HTML¹ eingebettet werden kann und bei manchen Web-Browsern (Netscape, Internet-Explorer) die Ausführung von bestimmten Funktionen und Aktionen innerhalb des Web-Browsers bewirkt.

Im Gegensatz zu Java ist JavaScript

- keine selbständige Programmiersprache,
- nicht von der Browser-Version unabhängig,
- nicht mit den notwendigen Sicherheitsmechanismen ausgestattet.

Java Development Kit

Das JDK² umfasst die für die Erstellung und das Testen von Java-Applikationen und Applets notwendige Software, die Packages³ mit den zur Grundausstattung gehörenden Java-Klassen, und die Online-Dokumentation.

Zur Software gehören der Java-Compiler, das JRE⁴ (die Java Virtual Machine) für die Ausführung von Applikationen, der Appletviewer für die Ausführung von Applets, ein Java-Debugger und verschiedene Hilfsprogramme.

Die Online-Dokumentation umfasst eine Beschreibung aller Sprachelemente und aller Klassen des API.

Java ist eine relativ junge Programmiersprache und daher noch immer in Entwicklung, d.h. es kommen immer wieder neue Versionen mit Ergänzungen und Verbesserungen heraus: Die Urversion ist JDK 1.0 (1995).

Im Jahr 1997 kam Version 1.1 heraus, das brachte sowohl Änderungen und neue Konzepte (bei den Methodennamen gemäß den Beans-Konventionen, beim Event-Handling, bei den Text-Files) als auch Erweiterungen durch zusätzliche Packages (z.B. für Datenbanken).

Ende 1998 kam Version 1.2 mit umfangreichen Erweiterungen (unter anderem Swing).

Anfang 2000 folgte Version 1.3 mit Fehlerkorrekturen und Performance-Verbesserungen.

Seit Februar 2002 gibt es die Version 1.4, die wieder ein paar größere Erweiterungen, vor allem in der XML Unterstützung, mit sich brachte.

Seit Version 1.2 wird das JDK auch "Java-Plattform 2" genannt, und das "Java 2 SDK⁵" bedeutet dann das jeweils aktuelle JDK 1.x, unterteilt in:

- J2SE = Java 2 Standard Edition (mit Graphischen User Interfaces)

¹ HTML: HyperText Markup Language

² JDK: Java Development Kit

³ Packages: hier: Softwarepakete

⁴ JRE: Java Runtime Environment

⁵ SDK: Software Development Kit

- J2EE = Java 2 Enterprise Edition (mit Server-seitigen Extensions)
- J2ME = Java 2 Micro Edition (für kleine mobile Geräte)

Die Browser-Unterstützung für Java-Applets hinkt dieser Entwicklung meistens um 1 bis 2 Jahre nach:

Ältere Web-Browser unterstützen (wenn überhaupt) nur Version 1.0. Die meisten Browser-Versionen unterstützen wenigstens teilweise Version 1.1 (Netscape ab Version 4.06, Internet Explorer ab Version 4.0).

Mit Hilfe des "Java Activator" (Java Plug-in) von der Firma Sun kann man auch ältere Browser-Versionen auf die jeweils neueste Java-Version (1.2 etc.) aufrüsten. Bei Netscape ab Version 6.0 und Mozilla und bei Internet Explorer ab Version 6.0 ist dies der Standardfall.

Das JDK für ein bestimmtes System erhält man meist kostenlos (z.B. zum Download über das Internet) vom jeweiligen Hersteller, also die Solaris-Version von Sun, die HP-Version von HP, die IBM-Version von IBM. Versionen für Windows-PC, Macintosh und Linux kann man von Sun oder IBM bekommen.

5.2.3 VCL

Eine der Aufgaben der Konfigurationsschnittstelle ist die Übergabe der Anweisungen an das Projekt im ECAD System. Diese unterstützten meist nur eine vom Systemhersteller zur Verfügung gestellte Makrosprache. Diese fest in das Variantemodul zu integrieren würde aber das Modul an ein bestimmtes ECAD-System binden, so dass die Einführung einer unabhängigen, neutralen Sprache ein enormer Schritt in Richtung Systemunabhängigkeit ist. VCL¹ bietet die Möglichkeit, Anweisungen in einem neutralem Format zu definieren. Diese müssen dann vom Schnittstellenmodul in die jeweilige Makrosprache des verwendeten ECAD-Systems übersetzt werden. Der Vorteil, der sich dadurch ergibt ist, dass man das Variantemodul an jedes beliebige ECAD-System ankoppeln kann, sofern eine Schnittstelle dafür implementiert wurde. Die Sprache selbst besteht nur aus einfachen Steueranweisungen mit der Möglichkeit, Funktions- und Produktgruppen für ein Variantenprojekt zu verwenden und diese mit bestimmten Parametern zu belegen.

VCL muss nicht mit eigener Intelligenz ausgestattet werden, da dies vom Konfigurator und dem Kopplungsmodul übernommen wird. Ferner existiert das Wissen in den Daten der Funktionsgruppen und Produktgruppen.

Folgende Spezifikation sollte VCL erfüllen:

- Name des Zielprojekts
- Namen der FG die im Zielprojekt verwendet werden sollen
- Werte der Parameter für die eingesetzten Funktionsgruppen

Dadurch ergeben sich folgende Aufgaben, die man mit Hilfe von VCL erfüllen müsste:

¹ VCL: Variant Control Language; im Rahmen des Forschungsprojektes „ECAD-Variantenmodul“ entwickelt

- Erzeugen eines neuen Zielprojekts
- Kopieren von Funktionsgruppen
- Belegung der Funktionsgruppen Parameter
- Generieren der Varianten

5.2.4 XML

Was ist XML?

XML ist eine Metasprache für die Definition eigener Auszeichnungssprachen ("Markup Language"). Der amerikanische Linguist und Philosoph Noam Chomsky vertritt die Meinung, dass alle menschlichen Sprachen im Grunde gleich sind, weil sie über eine gemeinsame, universelle Basis verfügen. So eine "Universalgrammatik" könnte XML für die Kommunikationssprachen in der Datenverarbeitung werden.

Was ist eine Auszeichnungssprache?

Die Beschreibung einer Komponente von der in Kapitel 1 vorgestellten Beispielkonstruktion könnte beispielsweise wie folgt aussehen:

„Motor_1 vom Hersteller ABB mit 380V, 1800U/min und 5kW, die Ident-Nummer ist XXXXXX“.

Der Mensch als Leser dieser Nachricht kann die semantischen Informationen leicht erkennen und verarbeiten: „Motor_1“ ist offenkundig die Komponentenbezeichnung, „ABB“ ist der Hersteller, „380V“ ist die benötigte Versorgungsspannung, „1800 U/min“ gibt die Drehzahl an und „5kW“ die Leistung.

Wie soll eine Maschine (oder Software) die Semantik der Wörter erkennen? Wir müssen ihr sagen, wie die einzelnen Wörter zu interpretieren sind. Dies kann zum Beispiel für optische Texterkennungssysteme durch Anordnung der Wörter auf der Seite (Formulare) oder durch Kennzeichnung der Wörter mitgeteilt werden. Die obige Beschreibung könnte dann so aussehen:

```
<Bauteil Identnr="XXXXXX">
  <Motor>
    <Hersteller>ABB</Hersteller>
    <Spannung Einheit="V">380</Spannung>
    <Drehzahl Einheit="U/min">1800</ Drehzahl >
    <Leistung Einheit="kW">5</ Leistung >
  </Motor>
</Bauteil>
```

Jetzt hat die Maschine alle Informationen, um die Nachricht verarbeiten zu können. Die sog. Marken (auf Englisch Tag) zeichnen die Teile des Dokumentes aus und ordnen ihnen damit eine Bedeutung zu. Die Maschine wird natürlich dadurch nicht intelligenter, die Marken lösen nur die einprogrammierten Verarbeitungsschritte aus.

Das ist XML, genauer gesagt: Das ist ein Beispiel für eine Sprache, die mit XML definiert wurde. Das Beispiel sieht syntaktisch wie HTML aus, was kein Zufall ist, wie die kurze Geschichte der XML zeigen wird.

Eine kurze Geschichte der XML

Ende der 60er Jahre haben Charles Goldfarb, Edward Mosher und Raymond Lorie bei IBM die Auszeichnungssprache GML¹ für Textformatierung entwickelt. Die Sprache wurde für die firmeninterne Dokumentation erfolgreich eingesetzt. Die Aufbereitung der Dokumente erfolgte, wie damals üblich, im Batch-Betrieb.

Etwa zur gleichen Zeit wurde bei der GCA²-Organisation ein Verfahren namens GenCode konzipiert, um generische Formatierungscodes für Satzsysteme verschiedener Hersteller definieren zu können.

Die beiden Technologien dienten als Grundlage (syntaktisch GML und semantisch GenCode) für die genormte verallgemeinerte Auszeichnungssprache SGML³. Die Normungsarbeiten begannen Anfang der 80er Jahre bei dem amerikanischen Normungsinstitut ANSI [28]. SGML wurde schließlich 1996 als ISO-Norm ISO2879:1986 [29] verabschiedet.

SGML gilt als eine sehr komplexe und umfangreiche Sprache (die Spezifikation ist über 500 Seiten lang) und blieb eine Sprache von wenigen Spezialisten. Dass die Konzepte von SGML richtig waren, bewies der Erfolg von HTML. Anfang der 90er Jahre hat Tim Berners-Lee am Kernforschungsinstitut CERN bei Genf die auf SGML beruhende Sprache HTML für die Präsentation und Verknüpfung von Dokumenten im Internet entwickelt. HTML ist inzwischen das erfolgreichste Format für elektronische Dokumente.

Das Internet bzw. World Wide Web war ursprünglich als Raum für Mensch-Mensch- bzw. für Mensch-Maschine-Kommunikation konzipiert. In der letzten Zeit wird jedoch das Internet auch als Medium für Maschine-Maschine-Kommunikation verwendet, was ganz andere Anforderungen an die verwendeten Kommunikationssprachen stellt.

Die Auszeichnungssprache HTML ist eine Beschreibungssprache für die Präsentation von Dokumenten. Der Schwerpunkt liegt auf der Präsentation, d. h. in einem HTML-Dokument werden die zu präsentierenden Daten und die Aufbereitungsanweisungen vermischt. Mit Hilfe der Darstellung und des Kontextes kann der Mensch die Semantik der Daten erkennen, eine Maschine (Software) kann jedoch keine semantischen Informationen erkennen. Ein anderes Manko von HTML ist, dass sie nicht anwendungsspezifisch mit eigenen Marken erweitert werden kann.

Im Jahr 1996 wurde beim W3C⁴-Konsortium [37] unter der Leitung von Jon Bosak, eines Systemarchitekten bei Sun, eine Arbeitsgruppe gegründet, um SGML "web-tauglich" zu machen. Das Ergebnis war eine ca. 30-seitige Spezifikation, die im Februar 1998 den Status einer "W3C-Recommendation" erhalten hat und den Titel "Extensible Markup Language (XML)" trug.

Die wichtigsten Entwurfzielen waren:

- XML soll zu SGML kompatibel sein.

¹ GML: Generalized Markup Language

² GCA: Graphic Communications Association

³ SGML: Standard Generalized Markup Language

⁴ W3C: World Wide Web Consortium

- XML soll einfach im Internet benutzbar sein.
- Die Anzahl der optionalen Merkmale soll möglichst klein sein.
- XML-Dokumente sollen leicht erstellbar und auch für Menschen lesbar sein.
- XML soll in möglichst vielen Anwendungsgebieten einsetzbar sein.

In der Terminologie der Auszeichnungssprachen wird eine in XML formulierte Beschreibung als XML-Dokument bezeichnet, auch wenn der Inhalt nichts mit Textverarbeitung zu tun hat. Bild (13) zeigt den "Stammbaum" von XML.

XML - Stammbaum

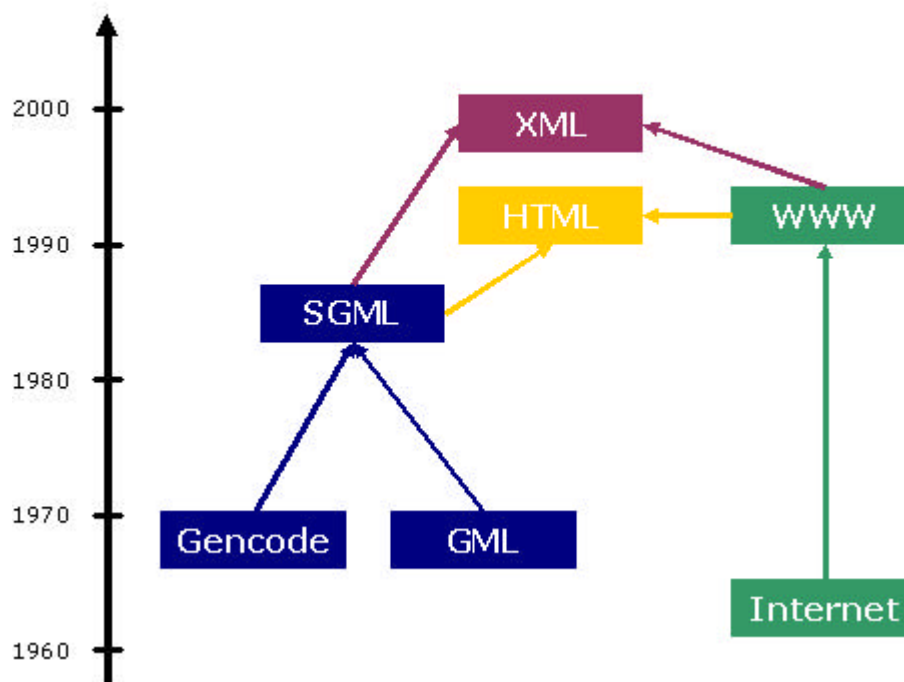


Bild (13) "Stammbaum" von XML

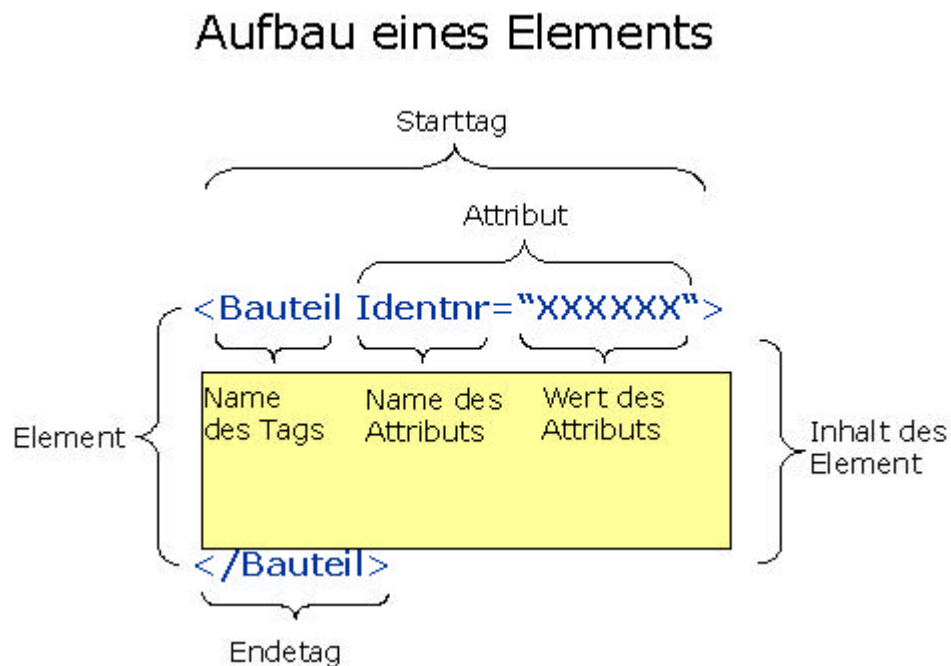
Anatomie von XML

Ein XML-Dokument besteht aus:

- einem Prolog mit einer "Startanweisung";

- einem Verweis auf die externe sog. DTD¹- und/oder einer internen DTD-Beschreibung (optional);
- einem Wurzelement;
- hierarchisch strukturierten Elementen;
- Verarbeitungsanweisungen (optional);
- Kommentaren (optional);
- sog. CDATA²-Abschnitten (optional);
- sog. Entitäten.

Aus Platzgründen wird nur auf die wichtigsten Teile eingegangen. Der Prolog Bild (14) besagt, welche XML-Version und optional welcher Zeichensatz verwendet wird:



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

Bild (14) Aufbau des Prologs eines XML-Dokuments

Der Inhalt besteht aus hierarchisch strukturierten Elementen, d. h. alle Elemente außer dem Wurzelement haben einen "Vater". Ein Element enthält eine Anfangsmarke, eine Endmarke und einen zwischen den Marken geklammerten Inhalt. Der Inhalt kann aus weiteren Ele-

¹ DTD: Document Type Definitions

² CDATA: Charakter Data

menten und aus reinem Text bestehen. Die Elemente können auch Attribute haben, die in der Anfangsmarke definiert werden.

HTML-Kenner würden sagen: "Das ist ja wie HTML!". Fast, es gibt einige wesentliche Unterschiede:

- Elemente dürfen sich nicht überlappen:
`<p>Das ist falsch</p>`
- Leere Elemente können in Kurzform benutzt werden:
`
` oder `</br>`
- Jedes Element muss mit einer Endemarke abgeschlossen sein:
`<p>Ein Paragraph</p>`
``
`
`
- Die Groß- und Kleinschreibung der Elementnamen und der Attributnamen muss beachtet werden.
- Attributwerte müssen in Anführungszeichen gesetzt werden:
`` oder ``

Woher weiß man, ob ein XML-Dokument die richtigen Elemente in der richtigen Reihenfolge enthält? Dazu dient die sogenannte DTD Bild (15). Die DTD beschreibt den Wortschatz der XML-Sprache (Elemente und Attribute) und die Regeln, in welcher Reihenfolge und wie oft die Elemente im Dokument vorkommen dürfen. Die DTD für das Beispiel mit dem Bauteil sieht folgendermaßen aus:

```
<!ELEMENT Hersteller (#PCDATA1)>
<!ELEMENT Bauteil (Motor)>
<!ATTLIST Bauteil
  Identnr CDATA #IMPLIED>
<!ELEMENT Motor (Hersteller, Spannung, Drehzahl, Leistung)>
<!ELEMENT Hersteller (#PCDATA)>
<!ELEMENT Spannung (#PCDATA)>
<!ATTLIST Spannung
  Einheit CDATA #IMPLIED>
<!ELEMENT Drehzahl (#PCDATA)>
<!ATTLIST Drehzahl
  Einheit CDATA #IMPLIED>
<!ELEMENT Leistung (#PCDATA)>
<!ATTLIST Leistung
  Einheit CDATA #IMPLIED>
```

¹ PCDATA: Parseable Character Data

Die DTD-Beschreibung kann vollständig innerhalb oder außerhalb des Dokumentes oder teils im Dokument und teils extern stehen.

Einen Parser, der die DTD-Beschreibung auswertet, um die Korrektheit eines XML-Dokumentes zu prüfen, nennt man validierenden Parser und das geprüfte Dokument bezeichnet man als gültig. Der XML-Parser muss aber nicht immer so streng sein. Wenn der Parser das Dokument nur auf Einhaltung der allgemeinen XML-Syntax prüft, ohne die Grammatik aus der DTD zu berücksichtigen, ist das Dokument nur wohlgeformt.

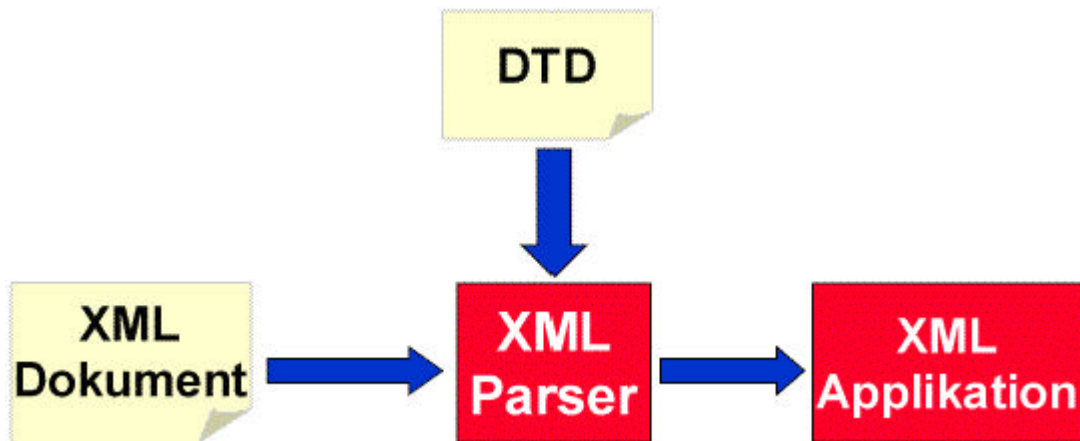


Bild (15) Ein validierender XML-Parser

Die XML-Familie

XML ist mehr als nur eine Metasprache: XML ist inzwischen ein Gattungsbegriff für eine ganze Reihe von Technologien Bild (16).

Die XML-Familie

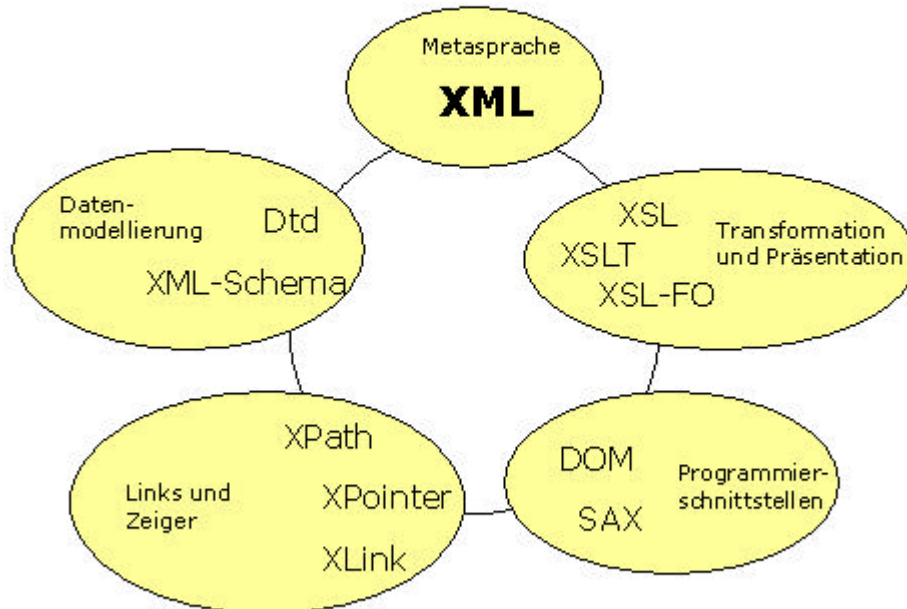


Bild (16) Die XML-Familie

Programmierschnittstellen DOM¹ und SAX²

Die Anwendungen, die XML-Dokumente verarbeiten, benötigen eine Programmierschnittstelle zum Parser. Es gibt zwei Arten von Schnittstellen: eine baum-orientierte und eine ereignis-orientierte Schnittstelle. Die baum-orientierte Schnittstelle DOM ist ein W3C-Standard und ermöglicht den Zugriff auf die gesamte geparsete Baumstruktur. Der Nachteil der Schnittstelle ist, dass das Dokument vollständig geparsert werden muss, bevor man auf den Inhalt zugreifen kann.

Um dieses Problem zu lösen, wurde in der XML-Gemeinde die ereignis-orientierte Schnittstelle SAX vorgeschlagen. Die Anwendung wird bereits während des Parsens des XML-Dokumentes über Ereignisse benachrichtigt, beispielsweise wenn eine Anfangsmarke, eine Endmarke usw. erkannt werden. So kann die Anwendung eine eigene interne Darstellung aufbauen oder Teile des Dokumentes ignorieren.

Präsentation - XSL³

Für die Präsentation ist das "Familienmitglied" XSL zuständig. XSL besteht aus zwei Sprachen: eine Transformationssprache und eine Formatierungssprache. Mit Hilfe der Transfor-

¹ DOM: Document Object Model

² SAX: Simple API for Xml

³ XSL: eXtensible Style Sheet Language

mationssprache (XSLT) wird ein XML-Dokument in ein anderes XML-Dokument transformiert. Die Umsetzung kann den alten Inhalt übernehmen, verändern oder vollständig ersetzen. Möchte man das XML-Dokument in einem Browser präsentieren, kann das XML-Dokument nach HTML umgesetzt werden. HTML ist sehr "browser-zentriert" und bietet zum Beispiel für Druckpräsentation kaum Unterstützung.

Aus diesen Gründen wird die Formatierungssprache "XSL-Formatting Objects" (FO) standardisiert, die mediumsunabhängige Aufbereitungsmöglichkeiten bietet. Für die Präsentation in einem Browser kann auch die aus der HTML-Welt bekannte CSS¹-Sprache verwendet werden. CSS bietet aber nicht die Möglichkeit, die Struktur des Dokumentes zu verändern, wie dies XSLT tun kann.

Verknüpfen - XLink und XPointer

Die aus HTML bekannten Verknüpfungen ("Hyperlinks") gehen nur in eine Richtung: von der Quelle zum Zieldokument. Die erweiterten Verknüpfungen (XLink) in XML gehen weiter: Sie können gleichzeitig mehrere Dokumente verknüpfen und sogar bidirektionale Verbindungen definieren. Die erweiterten Verknüpfungen müssen nicht einmal in den beteiligten Dokumenten stehen, in diesem Fall spricht man von sog. Out-of-line-Links.

HTML-Links verweisen entweder auf das gesamte Dokument oder nur auf eine Stelle im Dokument, man hat aber keine Möglichkeit, nur einen Ausschnitt zu adressieren. Mit dem erweiterten Zeiger (XPointer) versucht man in XML dieses Problem zu lösen. Ein erweiterter Zeiger kann absolut oder relativ die Bestandteile eines Dokumentes adressieren.

Die beiden Sprachen XLink und XPointer befinden sich noch in der Standardisierungsphase.

DTD vs. XmlSchema

Das Konzept der DTD stammt aus der dokumenten-orientierten Welt SGML. XML wird aber nicht nur als Auszeichnungssprache für Dokumente benutzt, sondern allgemein für Datenaustausch. Die DTD kennt aber nur wenige Datentypen, außer Aufzählungen für Attribute und Zeichenketten.

Hier soll die neue Sprache XmlSchema helfen. XmlSchema kann alles, was die DTD kann, kennt darüber hinaus die gängigen Datentypen (string, real, boolean, ...) und ermöglicht die Definition eigener Typen. Ein weiterer Vorteil der XmlSchema-Sprache ist, dass sie selbst in XML formuliert wird, sodass kein spezieller Parser wie für DTD erforderlich ist.

Leider ist die Standardisierung der XmlSchema-Sprache noch nicht abgeschlossen.

Wo kann XML eingesetzt werden?

Es gibt kaum ein Gebiet in der Datenverarbeitung, wo XML nicht eingesetzt wird. Die Beispiele reichen von Sprachen für elektronische Geschäftsabwicklungen bis zur Beschreibung von Wetterberichten. XML kann überall benutzt werden, wo kommuniziert wird. Die Einsatzbereiche von XML - genauer gesagt, der Einsatz der durch XML-spezifizierten Sprachen - können in drei große Kategorien eingeordnet werden:

- Auszeichnungssprache für Dokumente.
- Beschreibung von Metadaten.

¹ CSS: Cascading Style Sheets

- Datenaustauschformat für Kommunikation.

DocBook ist ein Beispiel für eine Auszeichnungssprache für technische Dokumentationen, die aus der SGML-Welt stammt. Beim W3C-Konsortium wird gerade HTML nach den strengen XML-Syntaxregeln reformuliert (XHTML), um HTML zu modularisieren und erweiterbar zu machen. WML, die Web-Sprache für die mobilen Geräten mit eingeschränkter Anzeigefähigkeit (Handy, PDA usw.), ist auch eine XML-Anwendung.

```
<wml>
  <card>
    <p>
      Please <i>select</i> your choices:
      <select iname="P" iname="I" ivalue="1;2" multiple="true">
        <option value="J">Java</option>
        <option value="CPP">C++</option>
        <option value="P">Python</option>
        <option value="V">Visual Basic</option>
      </select>
    </p>
  </card>
</wml>
```

Im Java-Server-Komponentenmodell EJB wird XML als Sprache für die Beschreibung der Metadaten (sog. Deployment-Deskriptor) eingesetzt. Ein anderes Beispiel ist XMI¹, ein von der OMG standardisiertes Austauschformat für objektorientierte Entwurfswerkzeuge. Kommunikationsprotokolle wie CORBA²-IIOP³ oder DCOM⁴ benutzen ein binäres Format für die Übertragung der Daten. Diese Protokolle sind für stark gekoppelte Systeme geeignet, nicht jedoch für lose gekoppelte Systeme, wie es beispielsweise das Internet ist. Für diese Systeme verspricht ein XML-basiertes Protokoll viele Vorteile: Textformate sind plattformunabhängig, man benötigt nur einen Parser für alle XML-Formate, man kann ein XML-basiertes Protokoll leicht erweitern, ohne dass der Kommunikationspartner dadurch "ins Schleudern" käme. Microsoft hat ein solches Protokoll für objektorientierte Kommunikation vorgeschlagen. Das SOAP⁵-Protokoll soll nach der Vision von Microsoft das Protokoll für die Interoperabilität zwischen verschiedenen objektorientierten Systemen werden.

10 Gründe die für den Einsatz von XML sprechen

- 1. XML ist ein offener Standard mit hoher Akzeptanz.
- 2. XML ermöglicht eine klare Trennung zwischen Daten und Präsentation.
- 3. XML ist textorientiert.

¹ XMI: XML Metadata Interchange Format

² CORBA: Common Object Request Broker Architecture

³ IIOP: Internet-Inter-Orb Protocol

⁴ DCOM: Distributed Component Object Model

⁵ SOAP: Simple Object Access Protocol

- 4. XML ist erweiterbar.
- 5. XML ist selbstbeschreibend.
- 6. XML ist internationalisierbar.
- 7. XML ist plattform- und programmiersprachenunabhängig.
- 8. XML ist maschinell verarbeitbar.
- 9. XML ist geeignet für langfristige Datenablage.
- 10 XML ist leicht transformierbar.

Fazit

XML ist eine Metasprache für die Beschreibung von Auszeichnungssprachen. Um XML einsetzen zu können, müssen Sprachen mit einem eigenen Vokabular und eigener Grammatik definiert werden. Der Erfolg von XML wird stark davon abhängen, ob es gelingt einheitliche branchen-spezifische Sprachen zu standardisieren ("Vertikalisierung" von XML). Sonst droht, trotz der gemeinsamen Metasprache XML, das Schicksal des Turms von Babel.

XML ist eine vielversprechende Technologie, alle Probleme der Software-Industrie wird sie jedoch nicht lösen.

6 Anwendungs-Demobeispiel

Als Beispiel für den Einsatz des Variantenmoduls wird hier eine klassische Erzeuger-Verbraucher Konstellation dargestellt. Als Erzeuger wird hier ein Generator, mit den Attributen: 380Volt, Hersteller: ABB, 30A, ausgewählt. Ergänzt wird die Funktionsgruppe durch die Verbindungen mit den Bezeichnungen „X1“ und „X2“. Hierbei handelt es sich um Klemmen, die zu der Komponente gehören. Die gesamte Schaltung bildet somit die Einheit „Generator_1“. Diese wird im ECAD-System erstellt und könnte so aussehen:

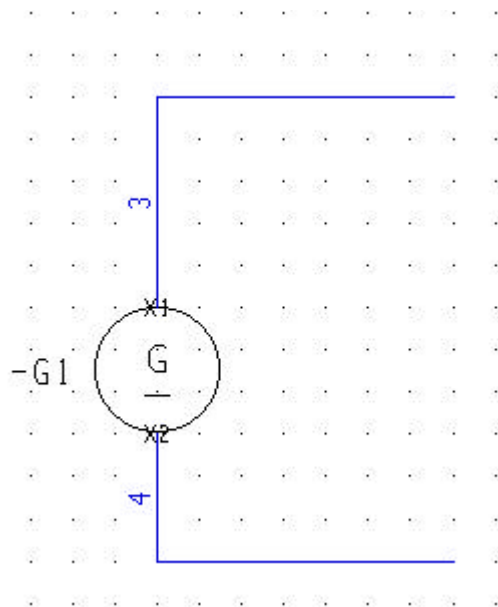


Bild (17) Generator_1

Die Schaltung muss zunächst durch den Benutzer in das Variantenmodul exportiert werden. Dabei wird die Schaltung mit der Export Funktion in ein verfügbares Datenaustauschformat transferiert. Der Vorteil, der sich hieraus ergibt, ist, dass man ein beliebiges ECAD-System, welches das Datenaustauschformat unterstützt, zum Erstellen der Schaltpläne benutzen kann. Man kann sogar vorhandene Schaltpläne verwenden. Im günstigsten Fall wäre diese Datei bereits im STEP-Format, denn dann würde eine Konvertierung des Austauschformates ins STEP entfallen. Die Datei wird dann vom Interfacemodul eingelesen. Dabei übersetzt das Interfacemodul die Daten gegebenenfalls in das STEP-Format und schreibt diese wiederum in eine neue Datei.

Diese Datei (STEP Format) kann nun vom Kopplungsmodul eingelesen werden. Die Funktionen, die das Kopplungsmodul bei diesem Vorgang durchzuführen hat, sind:

- Prüfung, ob ein Objekt mit der gleichen Bezeichnung bereits in der Datenbank existiert. Falls ja, aktuelles Objekt umbenennen.
- Speichern des aktuellen Objekts in die Datenbank

- Update an den Konfigurator schicken

Das Update an den Konfigurator geschieht in der prototypischen Realisierung mittels Dateiaustausch. Das heißt, das Kopplungsmodul schreibt in eine Datei, welche Komponenten es in seiner Datenbank zur Verfügung hat. Diese Datei kann nun vom Konfigurator eingelesen werden. Ab diesem Zeitpunkt kann ein Benutzer im Konfigurator die neu erstellten Komponenten auswählen.

Bevor die Funktionsweise des Konfigurators erklärt wird, muss zuerst der Verbraucher konstruiert werden. Er muss auf die gleiche Weise, wie der Erzeuger, dem Kopplungsmodul bekannt gemacht werden. Als Verbraucher wird ein Beispielmotor ausgewählt. Seine Werte sind 380V, 1800U/min, 5kW. Der Hersteller soll ABB sein. Die Schaltung könnte so aussehen:

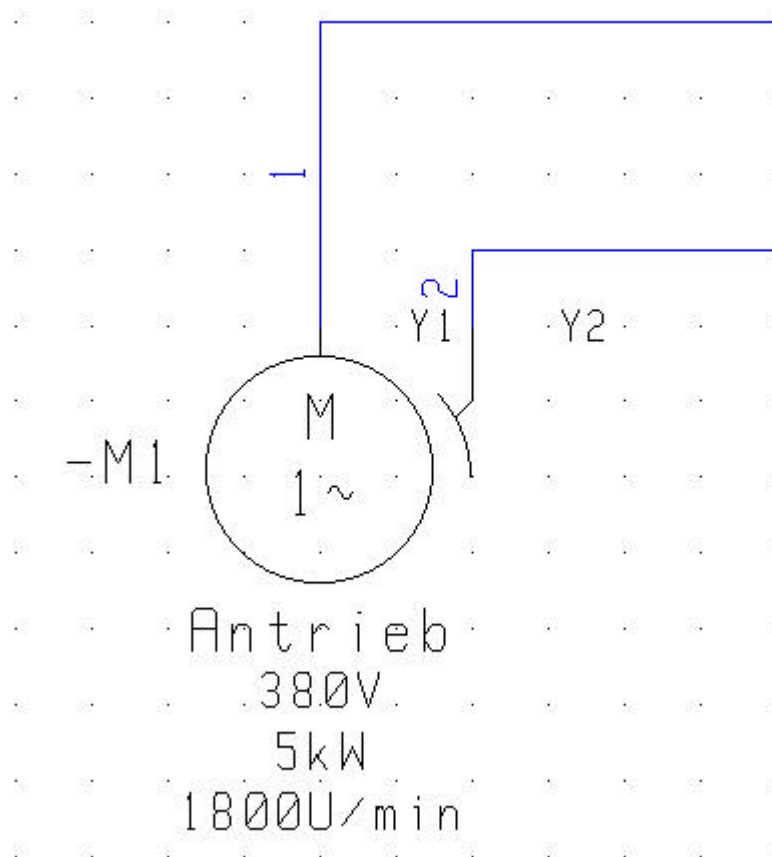


Bild (18) Motor_1

Um einen Einsatz von Varianten zu simulieren, müssen mehrere Komponenten zur Verfügung stehen. Daher muss anschließend ein weiterer Motor „Motor_2“ erstellt werden, der die Werte 380V, 10kW, 3600U/min hat. Der Hersteller dieses Motors soll die Firma Siemens sein. Die Schaltung dafür sieht folgendermaßen aus:

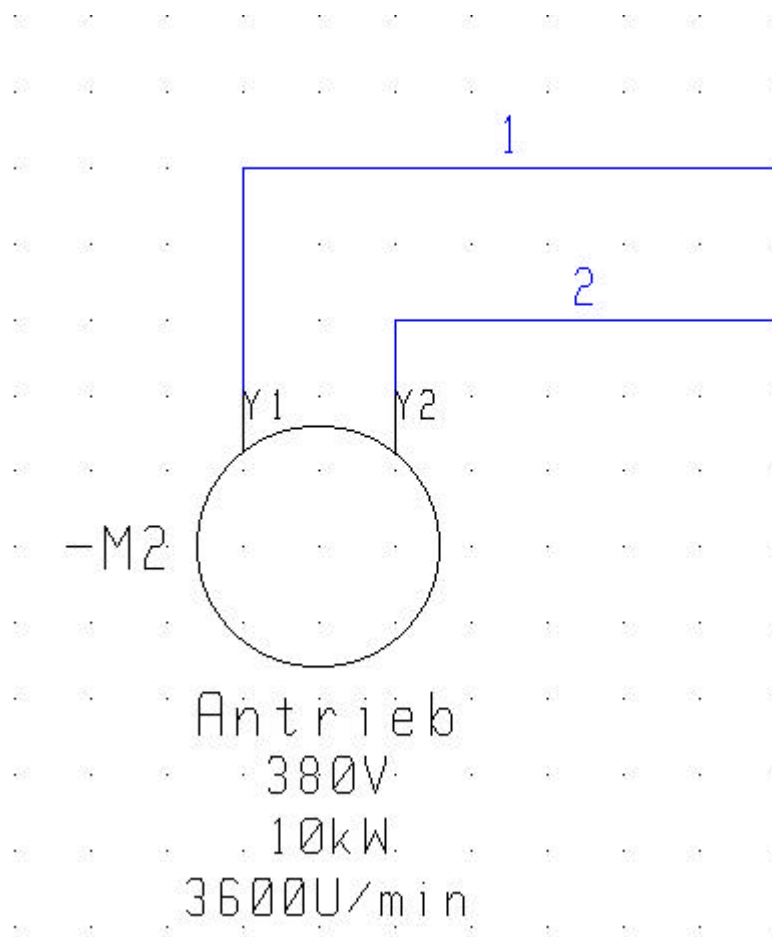


Bild (19) Motor_2

Das Aktualisieren des Konfigurators geschieht auf die gleiche Weise, wie beim „Generator_1“.

An diesem Punkt kann man davon ausgehen, dass die Komponenten „Motor_1“, „Motor_2“ und „Generator_1“ sowohl in der Datenbank des Kopplungsmoduls, als auch im Konfigurator zur Verfügung stehen.

Nun kann der Benutzer im Konfigurator ein neues Projekt anlegen. Er vergibt ihm den Name „Erzeuger-Verbraucher“. Auf der linken Seite des Konfigurators sollte sich eine Art Objektbrowser befinden (ähnlich dem Windows Explorer), mit dem der Benutzer nun zunächst auf die Klasse Generatoren klickt, worauf sich diese Kategorie öffnet und die verfügbaren Generatoren sichtbar werden. In diesem Beispiel steht hier lediglich der Generator_1 zur Verfügung. Dieser wird durch einen Doppelklick selektiert und in die Auswahlliste auf der rechten Seite des Konfigurators übernommen. Nun muss der Benutzer auf Motoren klicken, worauf sich ihm die Auswahl der verfügbaren Motoren öffnet. In diesem Beispiel kann er sich zwischen „Motor_1“ und „Motor_2“ entscheiden. Er übernimmt den gewünschten Motor (beispielsweise Motor_1) wiederum durch einen Doppelklick in die Auswahlliste.

Die Komponenten für das Erzeuger-Verbraucher Beispiel sind auf der rechten Seite des Konfigurators aufgelistet. Der Benutzer muss anschließend die Prüfung der Konfiguration anstoßen. Hier überprüft der Konfigurator primär, ob die Anschlüsse zueinander kompatibel sind. Im Beispielszenario besitzt der Motor_1 die Anschlüsse „X1“ und „X2“, welche zunächst mit

den Anschlüssen des „Generator_1“ verglichen werden. Das heißt, der Konfigurator überprüft, ob der „Generator_1“ ebenso wie der „Motor_1“ mit der Spannung 380V betrieben wird. Falls dies nicht zutrifft, meldet er dem Benutzer, dass diese zwei Komponenten so nicht konfigurierbar sind. Anderenfalls übergibt er diese Komponenten an das Kopplungsmodul. Des Weiteren muss er auch den Namen des neuen Projekts übergeben und eine Variable, die definiert, dass es sich um ein neues Projekt handelt. Diese Übergabe geschieht in dem prototypischen Beispiel wiederum durch Dateitransfer.

Das Kopplungsmodul empfängt die Namen der Komponenten und kann die dazugehörigen Daten aus der Datenbank importieren. Weiterhin werden die Verbindungen zwischen den einzelnen Komponenten erstellt, somit wird eine interne Verbindungsliste erzeugt. Nun muss das Kopplungsmodul Anweisungen generieren, mit denen ein neues Projekt erzeugt wird. Die einzelnen Anweisungen werden sequentiell in eine Datei gespeichert, so dass diese vom Interfacemodul verarbeitet werden können.

Nun sollte die neue Konstruktion in das ECAD-System wieder einfließen. Hier bieten sich drei Möglichkeiten an:

- Das Kopplungsmodul generiert die Daten in ein Format, das auch das ECAD-System unterstützt. Der Vorteil hierbei ist, dass man keine weitere Konvertierung der Daten vornehmen muss. Der Nachteil ist allerdings, dass man einen Parser in das Kopplungsmodul implementieren muss, damit dieser die Daten in dem gewünschten Format erzeugt. Ein weiterer negativer Aspekt ist, dass man das Variantenmodul auf diesen Standard festlegt, und somit nicht universell einsetzbar macht.
- Die zweite Möglichkeit ist, die Daten im bestehenden Format an das Interfacemodul zu übergeben, das diese dann in das jeweilige Format des ECAD-Systems transferiert. Durch Implementierung von weiteren Interfacemodulen könnte man das Variantenmodul für verschiedene ECAD-Systeme universell einsetzbar machen.
- Die dritte Möglichkeit besteht darin, die Daten in das Format des jeweiligen ECAD-Systems zu exportieren. Die Vor- und Nachteile gleichen denen von Punkt 1, mit einem noch schwerwiegenderem Nachteil: Im Normalfall hat man keine Einsicht auf die Speicherstruktur der Dateien, die von ECAD-System abgelegt werden. Aus diesem Grund wird diese Variante bei dem prototypischen Beispiel nicht in Betracht gezogen.

Das Interfacemodul für das jeweilige ECAD-System übersetzt die Anweisungen vom Kopplungsmodul in die Makrosprache des ECAD-Systems. Die Ausgabe des Interfacemoduls liefert eine ASCII-Datei, mit einer Sequenz von Makrobefehlen. Diese Datei kann nun durch das API¹ des ECAD-Systems eingelesen werden, um die Anweisungen dort auszuführen. Ebenso werden die Daten der Komponenten übersetzt und in Form von Makrobefehlen an das ECAD-System übergeben. Dieses zeichnet nun die neue Konstruktion anhand der empfangenen Daten.

¹ API: Application Programming Interface

6.1 Statische Konfiguration

Die Grundidee der statischen Konfiguration basiert auf dem Baukastenprinzip. Hier hat der Benutzer vorgefertigte Komponenten zur Auswahl, die er wahlweise in sein Projekt einfügen kann. Er kann aber auch neue Komponenten anlegen, die ihm dann zur Verfügung stehen oder bei Bedarf überflüssige Komponenten löschen.

Vorraussetzung für die statische Konfiguration ist allerdings, dass die Komponenten als separate, unabhängige und abgeschlossene Objekte repräsentiert werden. Dies bedeutet, dass eine Datenstruktur dahinter stecken muss, die eine Black-Box Funktionalität aufweist. Dem Benutzer sind lediglich die Schnittstellen nach außen und eventuelle Parameter bekannt. Die interne Datenstruktur ist verborgen. Die Überprüfung auf Kompatibilität geschieht anhand der Schnittstellen. Passen alle Anschlüsse der ausgewählten Komponenten zusammen, können sie zu einem Bauteil zusammengefasst werden. Dieses wiederum ergibt eine neue Komponente. Der Konfigurator basiert nicht auf einem Wissenssystem, d.h. er kann in diesem Fall keine selbständigen Entscheidungen treffen. Dies würde bereits die Aufgabe der dynamischen Konfiguration (siehe unten) spiegeln.

6.2 Dynamische Konfiguration

Die dynamische Konfiguration lässt sich am besten an hand eines Beispiels aufzeigen. Nehmen wir an, ein Konstrukteur möchte eine Aufzuganlage für ein Kaufhaus entwerfen. Er hat vor geraumer Zeit bereits ein ähnliches Projekt erstellt, bei dem aber das Gebäude um drei Stockwerke kleiner war. Um Zeit und damit auch Kosten zu sparen, möchte er gerne die Konstruktionspläne des vorhergehenden Projektes wieder verwenden.

Die dynamische Konfiguration bietet ihm die Möglichkeit, sein bestehendes Projekt zu laden, dort über Parameterwerte Einstellungen zu verändern, und die Konstruktion anschließend als neues Projekt zu speichern. In dem Beispiel öffnet der Konstrukteur sein altes Projekt im Konfigurator, worauf die einstellbaren Parameter angezeigt werden. Einer der Parameter (ETAGEN) gibt die Anzahl der Stockwerke an. In seinem alten Projekt steht hier die Zahl „2“. Er verändert diese Einstellung auf „5“. Die Aufgabe des Konfigurators besteht nun darin, die gesamte Konfiguration automatisch so abzuändern, dass das neue Projekt komplett erstellt wird. Damit kann sich der Konstrukteur die neuen Schaltpläne, Stück- und Verbindungslisten anzeigen lassen, oder sie wahlweise ausdrucken. Hierzu müsste der Konfigurator nicht nur die Anschlüsse auf Kompatibilität überprüfen, sondern auch gegebenenfalls Bauteile ersetzen. In diesem Beispiel würde der Motor, mit dem der Aufzug im alten Projekt betrieben wurde, nicht ausreichen, um die neue Anlage anzutreiben. Deshalb müsste der Konfigurator diesen durch einen stärkeren Motor automatisch ersetzen, wobei er alle notwendigen Verbindungen und Probleme selbst auflösen müsste.

Um diese Funktionalität zu realisieren, müsste der Konfigurator wissen, welche Bauteile mit welchen Parametereinstellungen in Verbindung stehen. Diese Information müsste dem Konfigurator beim Erstellen der betreffenden Komponente bekannt gemacht werden. So könnte eine Einstellung der Parameter in einer programmierähnlichen Syntax lauten:

```
IF ($ETAGEN < 3) THEN
  SET MOTOR="Motor_1"
ELSIF (($ETAGEN > 3) AND ($ETAGEN <7)) THEN
  SET MOTOR="Motor_2"
ELSE
```

```
SET MOTOR="Motor_3"  
FI
```

Der Wert der Variablen ETAGEN ist abhängig von der gewählten Einstellung im Konfigurator. Natürlich müsste mit anderen Bauteilen analog verfahren werden. Es könnte sein, dass andere Kabel verwendet werden müssen, oder dass die Befestigungen angepasst werden müssen, usw.

Der automatische Wechsel des Motors zieht jedoch weitere Änderungen nach sich. Der neue Motor könnte mit mehr Spannung betrieben werden, so dass man auch die vorhandenen Sicherungen der Konstruktion ersetzen müsste. Dies sollte in der dynamischen Konfiguration ohne den Eingriff des Konstrukteurs erfolgen. In einer halbautomatischen Realisierungsvariante könnte der Konfigurator den Benutzer zur Interaktion zwingen. Dort sollten alle möglichen Varianten zur Auswahl stehen.

6.3 Realisierungsaspekte

Da für die dynamische Konfiguration ein immens hoher Programmieraufwand zu erwarten ist, sollte man sich bei einer prototypischen Realisierung auf die statische Konfiguration beschränken. Die Entwicklung einer dynamischen Funktionsweise, wie oben beschreiben, könnte aber, aufbauend aus den Erkenntnissen des Forschungsprojekts, von einer Softwarefirma durchaus realisiert werden können. Dazu sollte die hier entwickelte Datenstruktur eine Erweiterungsmöglichkeit bieten, oder bereit über diese Funktionalität verfügen.

6.4 Mögliche Probleme und Risiken

Selbst bei einer statischen Konfiguration müssen einige Probleme bedacht werden, die entweder durch das Variantenmodul oder aber durch das ECAD-System gelöst werden sollten. Die einzelnen Komponenten besitzen beispielsweise jeweils eine BMK, die im Falle einer Neukonfiguration neu vergeben werden müsste, falls es zu einer Namenskollision mit einer anderen BMK kommen sollte. Hierzu ist es notwendig, eine Liste aller BMKs mit den entsprechenden Querverweisen zu führen, so dass bei Bedarf geprüft werden kann, ob eine BMK bereits vergeben ist. Trifft dies zu, muss für die neue Komponente eine neue BMK generiert werden. Eine mögliche Realisierung hierfür wäre eine fortlaufende Nummerngenerierung.

Ein weiterer kritischer Punkt ist die Potenzialvergabe. Alle Verbindungen besitzen in der Regel Potentiale, die bei einer Zusammenführung von zwei oder mehreren Komponenten zunächst geprüft, und gegebenenfalls abgeglichen werden müssen. Man betrachte folgendes Beispiel:

Eine Komponente besitze zwei Verbindungen mit den dazugehörigen Potentialen „1“ und „2“. Potential „1“ hat den Wert 220V. Nun soll eine zweite Komponente an die erste angeschlossen werden, welche die Potentiale „3“ und „4“ besitzt. Bei dem Potential „3“ handelt es sich auch um eine Verbindung mit 220V. Das System muss genau dies erkennen, die Verbindungen „1“ und „3“ zusammenführen und der neuen Verbindung ein neues, eindeutiges Potential zuweisen. Es könnte zum Beispiel die „1“ als neues Potential wählen, sofern diese nicht schon vergeben wurde.

Auch hier könnte das Problem dadurch gelöst werden, dass man eine Liste der benutzten Potentiale führt, um ein eventuelles Doppelvorkommen zu eliminieren.

Die Lösung der oben genannten Probleme sollte im Variantenmodul durch das Kopplungsmodul durchgeführt werden. Der Konfigurator besitzt nicht die notwendigen Daten, um beispielsweise eine Neuvergabe der BMKs durchzuführen. Das Kopplungsmodul jedoch hat über seine Datenbank Zugriff auf alle in der Schaltung existierenden Komponenten.

7 Schlussfolgerung und Ausblick

Zusammenfassend ist zu sagen, dass die prototypische Realisierung eines kompletten Variantenkonstruktionsmoduls den zeitlichen Rahmen, der bei studentischen Arbeiten zur Verfügung steht, bei weitem sprengen würde. Hier muss man sich auf die Realisierung einer von der Funktionalität her eingeschränkten Version beschränken, in der das Potential und die Möglichkeiten, die in einem Variantenkonstruktionsmodul liegen, anhand einfacher Beispiele aufgezeigt werden. Diese erste prototypische Version soll dann das Interesse der Anwender und der ECAD-Systemhersteller wecken, und somit als möglicher Ansatz zur Weiterentwicklung der bestehenden ECAD-Systeme dienen.

In der Studienarbeit wurde zunächst ein Überblick über die Probleme heutiger ECAD-Systeme wiedergegeben, um dann die Thematik der Variantentechnologie zu erklären. Auf eine kurze Vorstellung des bestehenden Konzepts folgte dann eine Grobspezifikation des Neuentwurfs. In den folgenden Kapiteln wurde auf die Entwicklungsumgebung eingegangen. Die notwendigen Grundlagen und Methoden der Informatik wurden hier vermittelt, um eine Realisierung des Konzeptes durchführen zu können. Dazu zählen sowohl Grammatiken, welche dazu benötigt werden, um die Sprache VCL formal beweisen zu können, als auch die Grundlagen des Compilerbaus, die man braucht, um einen Parser, der als Schnittstelle zwischen dem ECAD-System und dem Variantenmodul dient, zu erstellen. Ferner wurden moderne Werkzeuge und Programmiersprachen (C++, Java) vorgestellt, mit denen die Implementierung vorgenommen werden könnte. Außerdem gibt das Kapitel 4.3 über Software Engineering eine neutrale Anleitung zur Vorgehensweise bei der Realisierung des Konzeptes.

Das bestehende Konzept musste so verändert werden, dass eine prototypische Implementierung im Rahmen einer Diplomarbeit ermöglicht wird. Zum einen wurden Aspekte nicht berücksichtigt, wie z.B. Versionsmanagement, zum anderen musste man sich darauf beschränken, die einzelnen Module so zu entwerfen, dass der Realisierungsaufwand bei höchstens einem Mannjahr liegt. Unter anderem wurden die Kopplungsmodule 1 und 2 zusammengefasst und vereinfacht. Weiterhin ist der Einsatz von VCL als Anweisungssprache vorgesehen, wobei diese hier auf Basis von XML realisiert werden soll. Ein weiterer wichtiger Unterschied, den ein so realisiertes, herstellerunabhängiges Variantenmodul mit sich bringen würde, ist der Einsatz als Cross-Compiler (siehe Kapitel 4.2). Mit der Implementierung von weiteren Interfacemodulen bietet sich die Möglichkeit, Schaltungen im Format A in ein Format B zu transformieren.

Das überarbeitete Konzept besteht nun aus 3 Modulen, dem Konfigurator, Kopplungsmodul 1 und dem Interfacemodul. Der Konfigurator bietet dem Benutzer die Möglichkeit, vordefinierte Varianten von Komponenten auszuwählen und in ein neues Projekt einzufügen. Dabei wird die Konsistenz der Schaltung geprüft, das heißt, ob die Bauteile miteinander kombinierbar sind. Eine in sich schlüssige Schaltung wird an das Kopplungsmodul 1 weitergegeben, welches die Verbindungen zwischen den einzelnen Bauteilen erstellt. Das Kopplungsmodul 1 nimmt eine Prüfung der BMKs und Potentiale der Schaltung vor. Danach muss das es eine Anweisungsliste generieren, die an das Interfacemodul weitergereicht wird. Dieses übersetzt die Anweisungen in die jeweilige Makrosprache des ECAD-Systems. Die Übergabe an das ECAD-System erfolgt sequentiell, in einer fest definierten Reihenfolge.

Vom momentanen Stand der Dinge aus betrachtet wird es wohl noch einige Zeit dauern, bis die ersten ECAD-Systeme, die eine vollwertige Variantenkonstruktion unterstützen, auf den

Markt kommen. Ein möglicher Grund dafür sind die mit der Entwicklung und Realisierung verbundenen hohen Kosten, da man von einem Aufwand von mehreren Mannjahren ausgehen muss. Ist der Stein dann aber erst einmal ins Rollen gekommen, kann es sich keiner der ECAD-Systemanbieter leisten, nicht nachzuziehen und auch eine Variantenkonstruktion anzubieten, weil der Kundenwunsch schon längst vorhanden ist, und dann eine Abwanderung der Kundschaft zu befürchten ist.

8 Literatur

- [1] Schäfer, D. ; Roller, D.: „Elektro-CAD am Wendepunkt“, IWT Magazin Verlag, Vaterstetten, 1999, Nr. 5, S. 36-38
- [2] Schäfer, D.: „Variantentechnologie für Elektrotechnik-CAD“, Universität Stuttgart, intern, unveröffentlichter Forschungsbericht, 2000
- [3] Schäfer, D. ,Roller, D.: „Elektrotechnik CAD“, Shaker Verlag, Aachen, 2001, ISBN 3826590422
- [4] Roller, D. ,Schäfer, D.: „Parametrische Modellierung – Situationsanalyse und Trends“, CAD-CAM REPORT, Dressler Verlag, Heidelberg, 1998, Nr. 5, S. 96-104
- [5] Roller, D.: „CAD – Effiziente Anpassungs- und Variantenkonstruktion“, Springer, Berlin, 1995, ISBN 3540587799
- [6] Roller, D. und 4 Mitautoren: „Kosten- und Zeitreduzierung in der Elektrokonstruktion und Anlagenprojektierung“, Expert Verlag, Renningen-Malmsheim, 1998, ISBN 3816916163
- [7] Roller, D. ; Schäfer, D. ; Richert, U.: „Variantentechnologie in Elektrokonstruktion und Anlagenprojektierung“, CAD-CAM REPORT, Dressler Verlag, Heidelberg, 1998, Nr. 9, S. 90-97
- [8] Gesellschaft für Informatik e.V. (GI), <http://www.gi-ev.de/>
- [9] Camos Software und Beratung GmbH, <http://www.camos.de/>
- [10] Microsoft Corporation, <http://www.microsoft.com/>
- [11] Macromedia, <http://www.macromedia.com/>
- [12] Netscape, <http://netscape.com/>
- [13] Wegener, I.: „Theoretische Informatik“, B.G. Teubner, Stuttgart, 1993
- [14] Schöning, U.: „Theoretische Informatik – kurzgefasst“, Spektrum akademischer Verlag, Heidelberg,
- [15] Plödereder, E.: „Grundlagen der Programmiersprachen und Übersetzer Vorlesungsskript“, Fakultät Informatik der Universität Stuttgart, Stuttgart, WS 1998/99
- [16] Aho, A.V., Sethi, R., Ullmann, J.D.: „Compilers, Principles, Techniques, and Tools“, Addison-Wesley Publishing Company, 1988
- [17] Ludewig, J.: „Software Engineering Vorlesungsskript“, Fakultät Informatik der Universität Stuttgart, Stuttgart, 1997
- [18] IEEE: „Standards Collection Software Engineering“, IEEE, New York, 1993
- [19] Boehm, B. W.: „Software Engineering Economics“, Prentice-Hall, Englewood Cliffs, 1981
- [20] Humphrey, W.S.: „A Discipline for Software Engineering“, Addison-Wesley Publishing Company, 1995
- [21] Brändli, N.: „Standardisierung von Datenaustauschformaten“, CIM Management Vol.1, 1991
- [22] IGES Project, <http://www.nist.gov/iges/>

- [23] IGES/PDES Organisation: „The Initial Graphics Exchange Specification (IGES) version 5.1“, National Computer Graphics Association, Fairfax VA, 1991
- [24] AFNOR Z 68-300-1: „Standard for Exchange and Transfer. Introduction to SET“, AFNOR, Paris, 1989
- [25] Jäger, K.-W.: „Schnittstellen bei CAD-, CAE-Systemen: Grundlagen, Anwendungsbeispiele, Problematik, Lösungsansätze“, VDI-Verlag, Düsseldorf, 1991
- [26] DIN-Normen, Beuth Verlag GmbH, <http://www.beuth.de/>
- [27] Electronic Design Interchange Format (EDIF), <http://www.edif.org/>
- [28] American National Standards Institute (ANSI), <http://www.ansi.org/>
- [29] ISO, International Organisation for Standardisation, <http://www.iso.ch>
- [30] „Introducing STEP“, The Foundation for Product Data Exchange in the Aerospace and Defense Sectors, <http://strategis.ic.gc.ca/STEPguide>
- [31] ProStep, <http://www.prostep.com/de/>
- [32] Sun Microsystems, <http://www.sun.com/>
- [33] Silicon Graphics Inc., <http://www.sgi.com/>
- [34] Hewlett-Packard, <http://www.hp.com/>
- [35] IBM (International Business Machines), <http://www.ibm.com/>
- [36] Stroustrup, B.: “Die C++ Programmiersprache”, Addison-Wesley Publishing Company, 1992
- [37] The World Wide Web Consortium, <http://www.w3.org/>

9 Index

Ableitungsnotation	21	EDIF	42
Analyse beim Compilerbau	28	IGES	39
Analyse beim Software Engineering	35	SET	41
Anwendungsprotokoll 212	45	STEP	43
Betriebssysteme	48	VNS	42
C++	52	Probleme	72
Chomsky-Hierarchie	21	Realisierungsaspekte des Variantenmoduls	72
Codegenerierung beim Compilerbau	31	Rechtsableitung	20
Codeoptimierung beim Compilerbau	30	Risiken	72
Codierung beim Software Engineering	36	Semantische Analyse beim Compilerbau	29
Compiler	26	SET	41
Analyse	28	Simple LL(1) Grammatiken	25
Codegenerierung	31	Software Engineering	31
Codeoptimierung	30	Analyse	35
Lexikalische Analyse	28	Codierung	36
Semantische Analyse	29	Entwurf	36
Syntaktische Analyse	29	Software-Life-Cycle	32
Synthese	30	Spezifikation	35
Zwischencodeerzeugung	30	Test	37
Datenstruktur des Variantenmoduls	14	Software-Life-Cycle	32
Dynamische Konfiguration	71	Spezifikation beim Software Engineering	35
ECAD-Systeme	51	Standards	38
EDIF	42	EDIF	42
Entwurf beim Software Engineering	36	IGES	39
EXPRESS	44	SET	41
FIRST-Mengen	22	STEP	43
FOLLOW-Mengen	22	VNS	42
Grammatiken	20	Statische Konfiguration	71
Chomsky Typ-0	21	STEP	43
Chomsky Typ-1	21	Anwendungsprotokoll 212	45
Chomsky Typ-2	21	EXPRESS	44
Chomsky Typ-3	21	Syntaktische Analyse beim Compilerbau	29
LL(1)	25	Synthese beim Compilerbau	30
LR(k)	25	Test beim Software Engineering	37
Simple LL(1)	25	Typ-0 Grammatiken	21
IGES	39	Typ-1 Grammatiken	21
Interface des Variantenmoduls	18	Typ-2 Grammatiken	21
Java	53	Typ-3 Grammatiken	21
Komponentenkonstruktion	8	Variantenkonstruktionsmodul	13
Konfigurator des Variantenmoduls	15	Variantentechnologie	6
Kopplungsmodul	17	Komponentenkonstruktion	8
Lexikalische Analyse beim Compilerbau	28	Vorteile	7
Linksableitung	20	VCL	56
LL(1) Grammatiken	25	VNS	42
LR(k) Grammatiken	25		
Modulansatz	10		
Normen	38		

Vorgehensweisen bei der Realisierung des Variantenmoduls	19	Zwischencodeerzeugung	beim
XML	57	Compilerbau	30

Ich versichere, dass ich diese Arbeit selbständig verfasst und nur die angegebenen Hilfsmittel verwendet habe.

Ein Musterexemplar liegt bei der die Arbeiten entgegennehmenden Stelle zur Einsicht aus.