# Diplomarbeit

## Problemspezifikation für die Arbeitsplanerstellung rotationssymmetrischer Drehteile mit AutoCAD

*Andreas Niehaus*

April 1996

Betreuung: Prof. Dr. Michael M. Richter
Dipl. Inform. Frank Weberskirch

Arbeitsgruppe Künstliche Intelligenz
— Expertensysteme —
Prof. Dr. Michael M. Richter

Universität Kaiserslautern
Fachbereich Informatik

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Kapitel 1

# Einleitung

Um den Anforderungen am industriellen Markt gerecht werden zu können, sind Unternehmer gezwungen, immer komplexere, speziell auf den Kunden abgestimmte Produkte möglichst schnell in kleinen Losgrößen herzustellen. Dabei umfaßt die Herstellung eines neuen Produkts eine Vielzahl von Arbeitsschritten. Um den Marktanforderungen zu genügen wird bereits heute ein großer Teil dieser Arbeitsvorgänge computerunterstützt durchgeführt.

## 1.1 Computer Integrated Facturing (CIM)

Ein Ziel dabei ist es, eine vollständig integrierte Computerunterstützung für den gesamten Herstellungprozeß zu erreichen. Dieses Ziel fällt unter den Begriff des ***C**omputer **I**ntegrated **M**anufacturing (CIM)* [Roller, 1995; Encarnacão und Schlechtendahl, 1983]. Im Rahmen von CIM werden bisher besonders die Bereiche Konstruktion technischer Produkte durch Verwendung spezieller Entwurfswerkzeuge (***C**omputer **A**ided **D**esign, CAD*) und Fertigung durch computerbasierte Produktionsanlagen (***C**omputer **A**ided **M**anufacturing, CAM*) unterstützt. Bis heute gelang es aber nicht, den Bereich der Fertigungsplanung, der die Brücke zwischen CAD und CAM schlägt, adäquat zu automatisieren. In diesem Arbeitsschritt, dem sogenannten ***C**omputer **A**ided **P**rocess **P**lanning (CAPP)*, wird auf der Grundlage von entsprechenden Konstruktionsdaten ein detaillierter Arbeitsplan erstellt, nach dem das gewünschte Produkt gefertigt wird. Das Zusammenspiel zwischen CAD, CAPP und CAM, und das sich

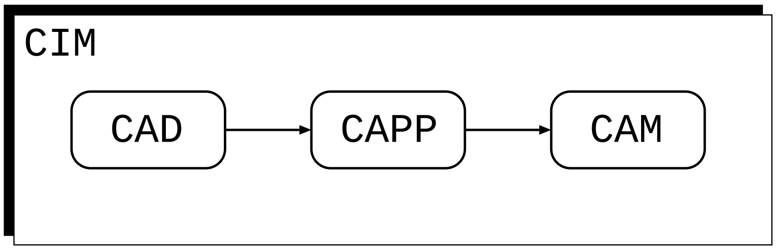


Abbildung 1.1: Computer Integrated Manufacturing (CIM)

daraus ergebende Konzept des CIM ist in Abbildung 1.1 skizziert. Mit Hilfe der Methoden der Künstlichen Intelligenz will man nun das Erfahrungswissen und das fertigungstechnische Wissen des Arbeitsplaners erfassen und für die automatische Verarbeitung nutzbar machen.

Für die vollautomatische Fertigungsplanung eines Werkstücks fordert der Planer ein bestimmtes Wissen und Informationen über das zu fertigende Werkstück. Gesucht ist also eine geometrische Darstellung eines Werkstücks und dessen Modellierung in Datenstrukturen, die für die Fertigungsplanung zwingend erforderlich sind, d.h. diese Datenstrukturen dienen dem nachfolgenden Planer als Eingabedaten. Das Planungsproblem besteht darin aus dieser Werkstückrepräsentation, welche die genaue Kontur des Werkstücks enthält, eine Sequenz von Bearbeitungs- bzw. Fertigungsschritten zu finden, die dieses Werkstück aus einem Rohteil fertigt. Die Ergebnisse des Planungsprozesses werden als Eingabedaten im Bereich des CAM verwendet. Dort werden *CNC-Maschinen (**C**omputer **N**umerical **C**ontrol)* benutzt, um auf Grundlage dieser Daten mit Hilfe verschiedener Aufspannungen und Fertigungswerkzeugen ein Rohteil in das gewünschte Werkstück zu überführen.

## 1.2 CAPlan

Die vorliegende Dipomarbeit ist Teil des Projekts *CAPlan (**C**omputer **A**ided **Plan**ning)* [Weberskirch, 1994; Weberskirch, 1995]. Im diesem Projekt geht es um die Entwicklung eines Planungssystems, welches auf die Unterstützung der Arbeitsplanerstellung in der computerintegrierten Fertigung für rotationssymmetrische Drehteile abzielt. Das Planungssystem CAPlan findet also innerhalb des CIM seine Anwendung im Bereich des CAPP.

CAPlan ist ein *domänenunabhängiges Aktionsplanungssystem.* Ausgehend von einer Anfangssituation in der zustandsorientierten Planungswelt ist eine Sequenz von Aktionen gesucht, die eine gegeben Anfangssituation in eine Zielsituation, dem Planungsziel, transformiert. Diese Sequenz von Aktionen nennt man den Plan. CAPlan abstrahiert von speziellen domänenspezifischen Gegebenheiten und ist durch diese *Domänenunabhängigkeit* leicht für verschiedenartige Planungsaufgaben einsetzbar, u.a. für die Arbeitsplanerstellung für rotationssymmentrische Drehteile. Das System stellt dazu dem Benutzer Verfahren zur Modellierung von speziellen Domänen zur Verfügung. Der Bereich der computerintegrierten Fertigung ist also nur als ein spezieller Anwendungsbereich zu sehen.

Wesentliche Grundlagen des Systems bilden der *SNLP-Ansatz* [McAllester und Rosenblitt, 1991] von David McAllester und David Rosenblitt, und das REDUX-System [Petrie, 1991] von Charles Petrie. CAPlan wurde aufbauend auf REDUX entwickelt, indem die Konzepte des SNLP-Ansatzes auf entsprechende REDUX-Konzepte übertragen wurden.

Zur Steuerung des Planungsvorgangs wurden verschiedene Kontrollkomponenten in das System integriert. Neben einer Kontrollkomponente, die die Standardstrategie von SNLP realisiert, wurde z.B. durch die Komponente ***C**ase-**b**ased **C**ontrol (CbC)* [Keane *et al.*, 1995; Muñoz-Avila und Weberskirch, 1996] die Möglichkeit geschaffen, mit Techniken des analogen und fallbasierten Schließens den Planungsvorgang zu steuern. Weitere integrierte Kontrollstrategien finden sich z.B. in [Kettnaker, 1995].

Viele Planungsalgorithmen und -systeme versuchen, ein Planungsproblem selbstständig und autonom, d.h. ohne Interaktion mit der Außenwelt, zu lösen. Bei der Lösung eines realen Planungsproblems, d.h. aus einer Domäne der realen Welt, ist eine Interaktion mit einem *Experten* von großem Vorteil. Nur er kennt alle relevanten Informationen bezüglich der Domäne, und kann diese sinnvoll in den Planungsprozeß und somit auch in die Lösung mit einbringen. Das CAPlan-System unterstützt eine solche interaktive Vorgehensweise im Sinne eines *Intelligenten Planungsassistenten.* Auf der einen Seite kann der Planer zwar selbständig und systematisch nach einer Lösung suchen, aber mittels der interaktiven Komponenten kann er auch vom Benutzer gesteuert werden.

## 1.3 Zielsetzung der Arbeit

Mit der vorliegenden Diplomarbeit wird ein Werkzeug bereitgestellt, das die Generierung einer Problemspezifikation für die Arbeitsplanerstellung rotationssymmetrischer Drehteile ermöglicht. Innerhalb des beschriebenen CIM befaßt sich die Arbeit also mit dem Bereich des CAD und der Erstellung von Konstruktionsdaten, die dem nachgelagerten Arbeitsplanungssystem CAPlan als Grundlage für einen detaillierten Arbeitsplan dienen.

Das enstandene Werkzeug erlaubt dem Benutzer die graphische Modellierung von realen Werkstücken und die Generierung der von CAPlan geforderten Werkstückrepräsentation. Die Domäne der Drehteile beschränkt sich auf rotationssymmetrische Werkstücke, die als Kombination von primitiven Komponenten (z.B. Zylinder) und Fertigungsfeatures (z.B. Abrundungen) zu sehen sind. Die geometrische Darstellung bzw. Erstellung eines Werkstücks und die Erzeugung von entsprechenden Datenstrukturen erfolgt mit Hilfe eines CAD-Systems und wird als von der eigentlichen Fertigungsplanung unabhängiger Vorgang betrachtet. Dies führt zu der Idee, das CAD-System und die nachfolgenden Planungsprozesse in einer Client-Server-Umgebung zu implementieren, um die geometrische Erstellung eines Werkstücks und die Fertigungsplanung auch auf verschiedenen Rechnern und von verschiedenen Personen durchführen lassen zu können. Als CAD-Basissystem wurde dafür das System AutoCAD der Firma AutoDesk ausgewählt. Aus dem Konzept, die Erzeugung einer Problemspezifikation als eigenständigen Prozeß zu sehen, ergibt sich der zusätzliche Vorteil, die generierte Problemspezifikation als Eingabedaten für verschiedene Fertigungsplaner zu nutzen. Das Prinzip der Erzeugung dieser Planungsinformationen bleibt gleich und die Daten können leicht den Anforderungen des entsprechenden Planers angepaßt werden.

## 1.4 Übersicht

- Kapitel 2
  In diesem Kapitel werden allgemeine CAD-Grundlagen erläutert und mögliche Modellierungskonzepte für CAD-Probleme vorgestellt.

- Kapitel 3
  Da AutoCAD als CAD-Basissytem für die geometrische Erstellung von Drehteilen ausgewählt wurde, erfolgt hier eine Einführung in das AutoCAD-System. Im Vordergrund steht dabei eine Überblick der vielfältigen Möglichkeiten dieses kommerziellen CAD-Systems. Insbesondere der Aspekt der Applikationsentwicklung, die das AutoCAD-System bietet, wird hier beschrieben.

- Kapitel 4
  Hier werden die in Kapitel 2 vorgestellten Ansätze konkretisiert. Dieses Kapitel erläutert die Werkstückrepräsentation, die dieser Diplomarbeit zugrundeliegt, und stellt die Benutzeroberfläche und Funktionalität der im Rahmen dieser Diplomarbeit entstandenen Applikation zur Erstellung von Drehkörpern vor.

- Kapitel 5
  Da AutoCAD und CAPlan jeweils auf verschiedenen Rechnern arbeiten können, werden sie in einer Client-Server-Umgebung auf der Basis des Betriebssystems UNIX installiert. Diese Umgebung, die entsprechende Schnittstelle und deren Benutzung werden beschrieben.

- Kapitel 6
  Eine Zusammenfassung dieser Arbeit und eine Bewertung des der Implementierung zugrundeliegenden AutoCAD-Systems erfolgt in diesem abschließenden Kapitel. Zusätzlich wird ein Vergleich zwischen dem im Rahmen dieser Arbeit implementierten Werkzeug und ähnlichen Ansätzen vollzogen.

Wie im folgenden noch näher beschrieben wird, wurde die Implementation der entstandenen Applikation zum größten Teil mit einer LISP-Variante durchgeführt. Für die Einbettung in eine Client-Server-Umgebung wurde auf die Programmiersprache C und Smalltalk, in der das System CAPlan implementiert ist, zurückgegriffen. Das Verständnis der Implementation setzt daher gründliche Kenntnisse dieser Sprachen voraus. Empfohlene Literatur sind z.B. [Goldberg und Robson, 1983; Mayer, 1995; Kernighan und Ritchie, 1990] und die Handbücher des AutoCAD-Systems.

# Kapitel 2

# CAD Grundlagen

Zur Erstellung eines Werkstücks auf der Basis eines CAD-Systems gibt es eine Vielzahl von zugrundeliegenden Darstellungskonzepten. Jedes dieser Konzepte hat in bezug auf bestimmte Zielsetzungen Vor- und Nachteile. In diesem Kapitel werden die verschiedenen Konzepte und ihre Stärken und Schwächen vorgestellt, unabhängig von einem konkreten CAD-System. Einige dieser Konzepte finden sich auch im System AutoCAD (siehe Kapitel 3), das bei dieser Diplomarbeit benutzt wird.

## 2.1 Allgemeines

In den Produktionsbereichen Konstruktion und Arbeitsplanung, insbesondere der Fertigungsplanung, wird eine Vielzahl geometrischer Daten verarbeitet, so daß der Erfassung, Speicherung und Verwaltung geometrischer Daten in der Konstruktion eine zentrale Rolle zukommt. Ähnliche Verhältnisse ergeben sich auch bei den Planungsprozessen, die der Konstruktion nachgeschaltet werden.

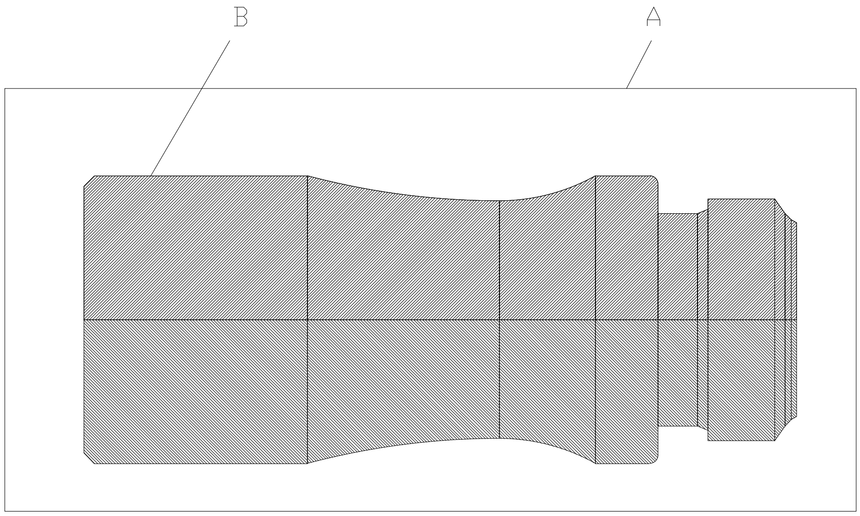


Abbildung 2.1: Regelbolzen A) Rohteilkontur B) Fertigteilkontur

Die in der Konstruktion und Arbeitsplanung verwendeten geometrischen Elemente, Daten und Informationen dienen in vielfältiger Form der Gestaltbeschreibung bzw. Gestaltveränderung des behandelten technischen Objektes. So muß für die Planung von Bearbeitungsaufgaben dem realen Objekt ein Ersatzmodell gegenübergestellt werden, das die Grundlage für den Planungsprozeß darstellt. Dieses Modell muß also Methoden zulassen, die der Lösung des Problems dienen. Abbildung 2.1 zeigt dazu beispielhaft den Querschnitt eines rotationssymmetrischen Bolzens. Das Beispiel zeigt, daß zur Lösung geometrisch orientierter Problemstellungen Verfahren der geometrischen Datenverarbeitung verwendet werden, die darin bestehen, daß sie geometrische Zusammenhänge in Form eines geeigneten Modells im Rechner abbilden und Werkzeuge bereitstellen, um das Modell interpretieren und modifizieren zu können [Spur und Krause, 1984].

## 2.2 Modellkonzepte

Ein Schwerpunkt der Rechnerunterstützung in Konstruktion und Arbeitsplanung ist das *geometrische Modellieren*. Dies resultiert aus der Erkenntnis, daß die überwiegende Anzahl der Aufgaben in diesen Bereichen nur unter Einbeziehung der Gestalt der betrachteten Objekte zu lösen ist. Informationen über die Geometrie der Objekte sind nicht nur zur Erzeugung von graphischen Darstellungen notwendig, sondern auch Eingangsgrößen für die nachgeschalteten Arbeitsprozesse.

Unter geometrischen Modellen werden Modelle verstanden, die neben der körperbeschreibenden Geometrie auch technologische Informationen und Zusammenhänge wiedergeben. Als geometrisches Modellieren kann man also den gesamten mehrstufigen Vorgang bezeichnen, ausgehend von der gedanklichen Vorstellung, dem Entwurf, bis hin zur Abbildung des vollständig gestalteten Produkts in einer rechnerinternen Darstellung.

Je nach Datenstruktur bzw. abgespeicherter Modellinformation unterscheidet man dabei zwischen Kanten-, Flächen- und Volumenmodelliersystemen [Roller, 1995].

### 2.2.1 Kantenmodell

*Kantenmodelle*, häufig auch *Drahtmodelle* genannt, werden dadurch beschrieben, daß ihre Eckpunkte gespeichert und die jeweiligen Körperkanten als geometrische Kurven repräsentiert werden. Häufig sind die dabei zulässigen Körperkanten auf Geradenabschnitte beschränkt. Solch ein Modell kann man sich aus dünnen Drähten zusammengebaut vorstellen. Diese Modelle haben aber mächtige Einschränkungen im Bereich der Analyse- und Simulationsmöglichkeiten und damit auch bei der Integration in die Fertigung. So lassen sich von Kantenmodellen keine schattierten Darstellungen ableiten, da die Flächen des Objektes nicht in der Datenstruktur festgehalten werden, sondern lediglich vom Menschen explizit zwischen geeigneten Kanten hineininterpretiert werden. Aus diesem Modell lassen sich keine Informationen extrahieren, wie z.B. Flächenbeschreibung oder Volumendaten, die als Grundlage für eine CNC-Maschine (**C**omputer **N**umerical **C**ontrol Maschine) dienen könnten. Dieses verhältnismäßig einfache Kantenmodell eigenet sich daher nur wenig für dreidimensionalen Geometrien. Seine Anwendung findet es eher bei zweidimensionalen Geometrien.

### 2.2.2 Flächenmodell

*Flächenmodelle* repräsentieren die Oberfläche eines Objektes oder Teile davon. Dieser Idee liegt die Auffassung zugrunde, daß technische Objekte von einer Oberfläche umgeben sind, die sie gegenüber der Umwelt abgrenzen. Diese „Haut“ eines Körpers wird geometrisch durch

Flächen dargestellt, die auch das Kernstück dieses Darstellungsschemas sind. Dabei wird üblicherweise für jede Fläche die Flächenberandung beschrieben. Somit ist das Flächenmodell in seiner einfachsten Form eine Erweiterung des Kantenmodelles, da sich durch Zusammenfassung bestimmter Kanten eine Fläche ergibt. Dem flächenorientierten Modell kann eine an der Mathematik orientierte Technik zugrundegelegt werden: Jede Fläche wird im Modell durch eine Anzahl ebener Flächen repräsentiert bzw. approximiert. Diese ebenen Flächen können Dreiecks- oder Rechteckflächen (allgemein Polyeder) sein (siehe Abbildung 2.2). Der Vor-

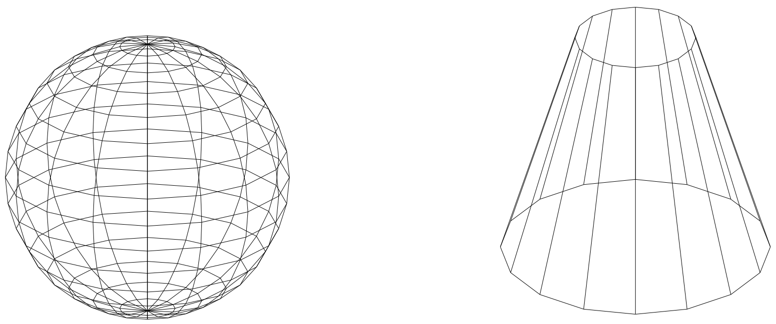

Abbildung 2.2: Geometrische Elemente approximiert durch ebene Flächen

teil dieser Technik liegt darin, daß die Flächen, die das Objekt approximieren, mit relativ einfachen mathematischen Methoden berechnet werden können. Ein Nachteil dieses Konzepts ist, daß die Maß- und Formtreue des Objektes natürlich abhängig ist von der Anzahl der zur Approximation jeder Fläche herangezogenen Polyederflächen. Eine Verbesserung der Approximation durch Erhöhung der Anzahl der Polyederflächen hat eine Erhöhung der zu verwaltenden Daten und Informationen in der rechnerinternen Darstellung zur Folge. Dies erhöht gleichzeitig auch die Verarbeitungszeit. Beispiele bei denen Flächenmodelle in der Regel Anwendung finden sind Außenhaut von Flugzeugtragflächen und von Automobilkarosserieteilen.

### 2.2.3 Volumenmodell

Die *Volumenmodelliersysteme*, auch *Solid Modelling Systems* genannt, zeichnen sich durch eine umfassende Modellrepräsentation aus. Am weitesten verbreitet ist die Beschreibung als *konstruktive Körpergeometrie*, englisch „*Constructive Solid Geometry*“ oder kurz *CSG* [Cunningham und Dixon, 1994; Zeid, 1991].

Beim CSG-Verfahren werden Objekte durch Grundkörper, auch *Primitive* genannt, und Verknüpfungsoperationen repräsentiert, die festlegen, wie die jeweiligen Primitive zu einem Ganzen zusammenzufügen sind. Genauer handelt es sich dabei um einen Binärbaum (siehe Abbildung 2.3), dessen innere Knoten Mengenoperationen (z.B. Vereinigung, Schnitt) beinhalten und dessen Blätter die eigentlichen Primitive darstellen. Als Primitive kommen hauptsächlich Zylinder, Kegel und Torus zum Einsatz. Komplexere Teilkonstrukte ergeben sich über volumentheoretische Operationen auf den bereits definierten Teilobjekten (siehe Abbildung 2.3). Eine explizite Beschreibung der einzelnen Körperkanten und -flächen liegt dabei aber nicht vor. Dennoch baut das Volumenmodell auf dem Flächenmodell auf, indem die das Objekt einhüllende Flächen zur Beschreibung des Volumens zusammengefaßt werden.

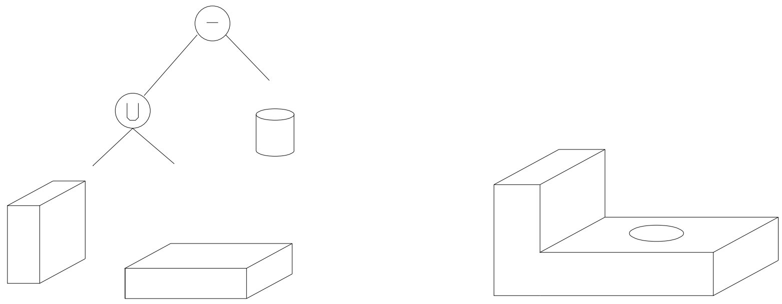


Abbildung 2.3: Volumenbildung durch mengentheoretisch definierte Basiskörper

## 2.3 Darstellungssysteme

CAD-Systeme lassen sich grob in zweidimensionale (2D) und dreidimensionale (3D) Systeme einteilen [Roller, 1995]. Von ihrer Philosophie her sind 2D-Systeme meist an die herkömmliche Konstruktion am Reißbrett angelehnt. 3D-Systeme dagegen beschreiben eine Konstruktion räumlich. Eine Mischform stellen dabei $2\frac{1}{2}$D-Systeme [Spur und Krause, 1984] dar. Die Vorgehensweise zur Beschreibung einer komplexen Gestalt eines Objektes ist dabei abhängig von dem zugrundeliegenden Modell. So hat jedes CAD-System in bezug auf mögliche Modellkonzepte, die in Abschnitt 2.2 vorgestellt wurden, Vor- und Nachteile. Dies wird in den nächsten Abschnitten erläutert.

### 2.3.1 2D-Systeme

Ihre Entwicklung begann schon in den frühen 70-er Jahren. Sie sind leicht zu erstellen und zu bedienen, die aufbauenden Algorithmen sind nicht allzu kompliziert, da ihre Konstruktionsvorschriften auf wenig komplexen Objekten wie Punkte, Kreise, Linien, etc. basieren. 2D-CAD-Modelle werden typischerweise in Form einer technischen Zeichnung repräsentiert. Im Gegensatz zu Zeichnungen, die mit einer allgemeinen Graphiksoftware erstellt werden, beinhalten CAD-Zeichnungen wesentlich mehr Informationen als nur eine graphische Darstellung in Form von Linien. Bei 2D-CAD-Systemen werden auch Informationen über die grundlegende Struktur der Konstruktion verwaltet. 2D-Systeme kommen besonders bei Erstellungen von Schaltplänen, z.B. bei der Elektronikindustrie, und Werkstattzeichnungen zum Einsatz [Roller, 1995]. Im 2D-Bereich ist eine Kanten- oder flächenorientierte Verarbeitung möglich.

Die *kantenorientierte* Verarbeitung (siehe Abschnitt 2.2.1) bedeutet, daß im System nur Konturelemente beschrieben werden können. Konturelemente sind die Eckpunkte des Objekts und die jeweiligen Kanten zwischen diesen Punkten. Die Darstellung des Objekts erfolgt nur durch Aneinanderreihen dieser Konturelemente. Da eine Definition des Bereichs zwischen zusammenhängenden Konturelementen nicht vorgenommen wird, ist es nicht ohne weiteres möglich, diese Bereiche automatisch zu schraffieren, denn Flächen kennt das kantenorientierte Konzept nicht.

Bei einer *flächenorietierten* zweidimensionalen Verarbeitung kann der Bereich zwischen geschlossenen Konturzügen zu einer ebenen Fläche definiert werden, da dieses flächenorientierte Konzept die Speicherung der Flächenberandung jeder Fläche vorschreibt. Dieses Verfahren bringt einige Vorteile mit sich. So kann jede Fläche einschließlich ihrer Berandung als eine Einheit angesprochen werden. Zudem sind automatisches Schraffieren und automatische Flächenverknüpfungen durch mengentheoretische Operationen wie Subtraktion, Addition, etc. möglich. Abbildung 2.4 zeigt ein Beispiel für die Anwendung der Flächenverknüpfung.

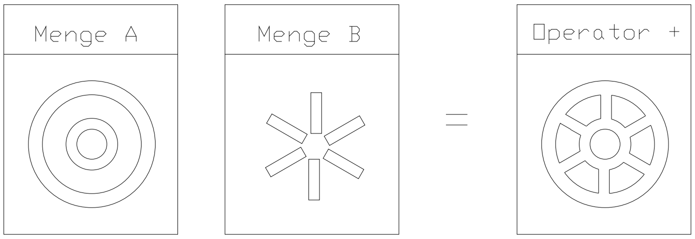


Abbildung 2.4: Prinzip der Flächenverknüpfung

Bei 2D-Systemen ergeben sich jedoch anwendungsabhängig Einschränkungen, die nur mit erhöhtem Aufwand kompensiert werden können oder auch nicht mehr akzeptabel sind.

2D-Modelle sind

- mehrdeutig, bereits bei der Konstruktion ist subjektive Interpretation erforderlich.
- schwierig zu analysieren, räumliche Zusammenhänge komplexer Konstruktionen sind nur schwer abzubilden, da der räumliche Aspekt bei kanten- bzw. flächenorientierter Verarbeitung durch Speicherung der Punkte bzw. Flächen nur simuliert wird, aber nicht wirklich Teil der Datenstrukturen ist.

Daher eignen sich 2D-Systeme für einige Anwendungsbereiche nicht. Dies gilt im besonderen für volumenorientierte Darstellungen.

### 2.3.2 3D-Systeme

Ganz allgemein haben 3D-Systeme gegenüber den 2D-Systemen die Fähigkeit, eine ganz andere Art von Objektdomäne darstellen bzw. gleiche Domänen adäquater darstellen zu können. So sind Objekte wie in Abbildung 2.5 in einem 2D-System nur unzureichend bzw. gar nicht darstellbar. 3D-Systeme sind aber in der Regel schwieriger zu bedienen, sie verwalten komplexere Datenmengen und verwenden völlig andere und weitaus aufwendigere Algorithmen. Für den wirtschaftlichen Einsatz speziell unter dem Aspekt der Wiederverarbeitung einmal erzeugter Modelle eignen sich aufgrund des Informationsgehalt dreidimensionale flächen- oder volumenorientierte Modelle [Spur und Krause, 1984].

Der Vorgang der Bauteildefinition in flächenorientierten 3D-Systemen kann folgendermaßen geschehen: Zunächst werden die Konturzüge, die das Bauteil ausmachen, definiert. In diesem Gerüst werden dann zwischen jeweils geschlossenen Konturzügen Flächen aufgespannt. Auf diese Weise werden Zusammenhänge zwischen den Kanten definiert. Mit Hilfe dieser Informationen ist es möglich, weitere räumliche Aspekte, wie z.B. Sichtbarkeitsverfahren, in die Darstellung von Objekten einzubinden. Abbildung 2.2 zeigt bereits Objekte, die auf der flächenorientierten Darstellung basieren.

Bei der Beschreibung eines Objekts durch Basisivolumenelemente muß eine andere Denkweise zugrunde gelegt werden. Das Bauteil wird gedanklich in die vom System bereitgestellten Basiselemente zerlegt. Die von den Systemen am häufigsten vordefinierten Basiselemente sind

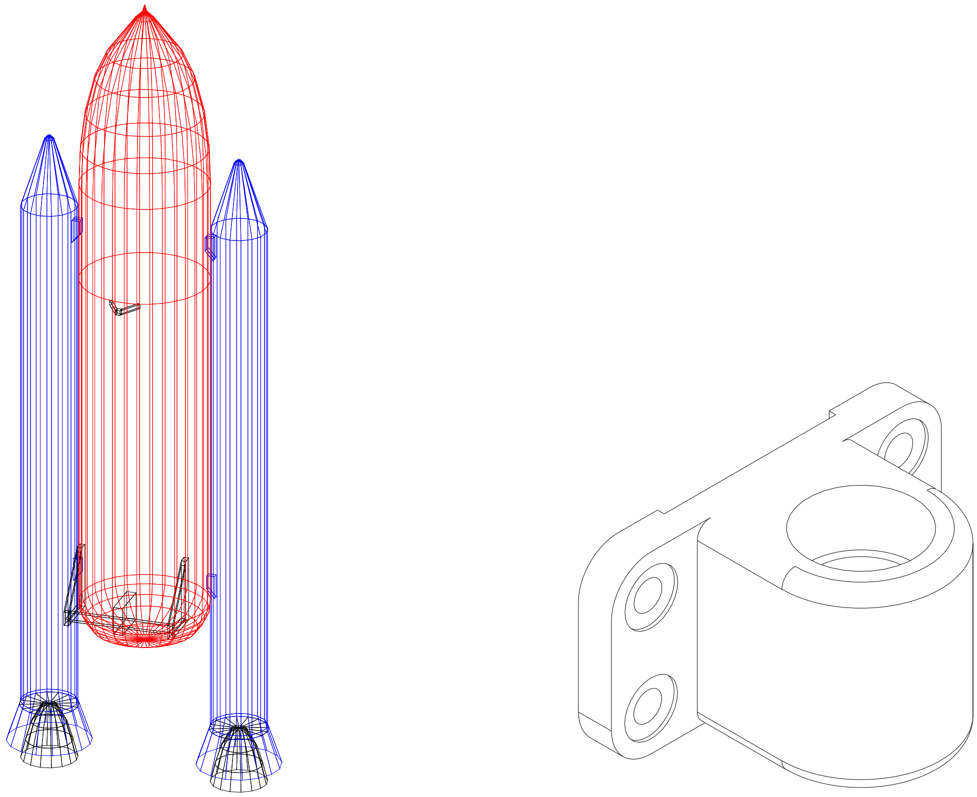

Abbildung 2.5: Komplizierte 3D-Objekte

Quader, Pyramiden, Zylinder, Kegel und Torus. Die Verknüpfung der einzelnen Grundkörper geschieht durch mengentheoretische Operationen, wie Vereinigung, Differenz und Durchschnitt. Zusätzlich kann der Benutzer weitere Volumenkörper erzeugen, die dadurch entstehen, daß begrenzte Flächen durch Verschiebung oder Rotation entlang einer Leitlinie bewegt werden. Durch diese Methode entsteht aus zweidimensionalen Grundkörpern eine ganz bestimmte Objektdomäne, die der *Regelkörper* bzw. *rotationssymmetrischen Drehkörper*. Zwei auf diese Art und Weise erstellten Objekte zeigt Abbildung 2.6.

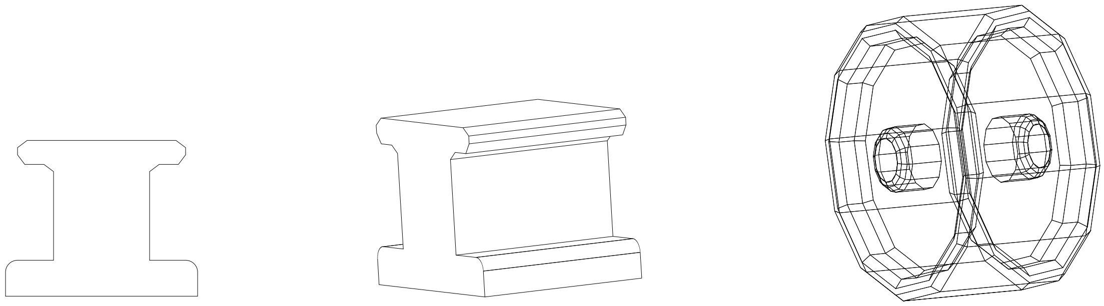

Abbildung 2.6: Verschiebung und Rotation eines 2D-Grundobjekts

### 2.3.3 $2\frac{1}{2}$D-Systeme

Bei einigen Applikationen, die auf 2D-Modellen aufbauen, darf der 3D-Aspekt nicht ganz außeracht gelassen werden, und muß bis zu einem gewissen Grad unterstützt werden. Systeme, die diese Idee berücksichtigen, heißen $2\frac{1}{2}$*D-Systeme* [Spur und Krause, 1984]. Hinter dem Begriff $2\frac{1}{2}$D steht dabei aber keine exakte Definition. Er deutet an, daß das 2D-Modell erweitert wird, aber dabei trotzdem nicht alle Aspekte der 3D-Geometrie betrachtet werden bzw. werden müssen. Die Erweiterung der 2D-Arbeitsweise besteht darin, daß man geschlossenen Konturzügen bzw. Flächen eine konstante oder variable Raumtiefe zuordnet. In Applikationen, die keine absolute 3D-Fähigkeit erfordern, ergeben sich so weniger komplexe und damit schnellere Algorithmen. Ein typischer Vertreter des $2\frac{1}{2}$D-Modells ist die Domäne der rotationssymmetrischen Drehkörper (siehe Abbildung 2.7).

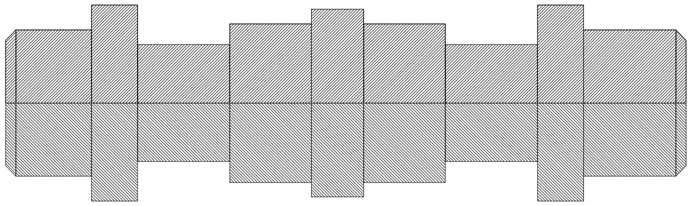

Abbildung 2.7: Rotationssymmetrischer Drehkörper

# Kapitel 3

# AutoCAD zur Entwicklung von CAD-Applikationen

CAD-Systeme können als Grundlage zur Entwicklung von problemspezifischen Applikationen dienen. Im Rahmen dieser Diplomarbeit wurde das System AutoCAD der Firma AutoDesk verwendet, welches eines der häufig verwendeten kommerziellen CAD-Systeme ist. Es stellt somit das CAD-Basissystem für aufbauende CAD-Applikationen dar. Das AutoCAD-Paket ist ein sehr mächtiges CAD-Werkzeug. Dabei unterscheidet es sich in primitiven Zeichenfunktionen, wie z.B. Linien ziehen, nicht besonders von anderen CAD-Systemen.

Die Grundideen des AutoCAD-Systems, die das System ausmachen und durch die es sich von anderen unterscheidet und hervorhebt, werden im ersten Teil dieses Kapitels vorgestellt. Die wichtigsten Konzepte werden danach detaillierter betrachtet und bewertet. Hauptaugenmerk liegt dabei auf den CAD-Konzepten, die die modelltheorethische Grundlage von AutoCAD bilden (siehe Kapitel 2) und auf der Entwicklung von problemspezifischen CAD-Applikationen auf Grundlage des AutoCAD-Systems.

## 3.1 Ideen von AutoCAD

Ziel der Firma AutoDesk war es, ein System zu entwickeln, daß in jeglichen Arbeitsbereichen, wo „gezeichnet" wird, seine Anwendung findet. Dabei kann natürlich nicht allen Anforderungen nachgekommen werden, da viele Firmen ihren eigenen Standard entwickelt haben, und jeder „Zeichner" seinen eigenen Arbeitsstil hat. Aus diesem Konflikt enstand der Gedanke einer *offenen Architektur*. Idee dieses Konzepts ist es, das zugrundeliegende System in einem gewissen Rahmen den individuellen Benutzerbedürfnissen anpassen zu können. AutoCAD kann also als Entwicklungsumgebung für problemspezifische CAD-Systeme benutzt werden. So können eigene *CAD-Applikationen* auf der Basis von AutoCAD erstellt werden, die dann den gewüschten Anforderungen genügen.

#### Menüs

Mit Hilfe programmierbarer Menüs läßt sich die AutoCAD-Oberfläche den Benutzerbedürfnissen anpassen. Die Änderung bereits vorhandener und Erstellung von neuen Menüs ist leicht und komfortabel. Die Benutzerhandhabung kann erhöht werden, indem zusätzliche Menüpunkte aufgenommen werden, die bestimmte Aufgaben symbolisieren. In dieser Art und Weise können mehrere eventuell komplexe Schritte mit einem einzelnen Menüklick ausgeführt werden. Durch programmierbare Menüs lassen sich professionelle Benutzeroberflächen für CAD-Applikationen entwickeln.

### Dialogfenster

Das Paket *Programmable Dialogue Box (PDB)* erlaubt es, eigene Dialogfenster zu implementieren, ähnlich zu denen, die durch AutoCAD bereits zur Verfügung stehen. Dabei stehen die Aspekte Layout und Verhalten bei der Erstellung von Dialogfenstern im Vordergrund. Die AutoCAD-Unterstützung für Dialogfenster ist dabei plattformunabhängig, d.h. das aktuelle Aussehen der Fenster ist abhängig vom *Graphical User Interface (GUI)* der benutzten Hardwareplattform. Somit brauchen auf unterschiedlichen Systemumgebungen keine Modifikationen am Layout oder Verhalten der Fenster vorgenommen werden. Die Möglichkeit der benutzerprogrammierten Dialogfenster verbessert Eingabe, Modifikation und Darstellung von Daten und orientiert sich stark an den Fenstersystemen X-Windows und Open-Windows.

### Prototypen

Der Benutzer kann eine sogenannte *Prototypzeichung* auswählen, die als Grundlage für eine neue Zeichnung dient. Ziel dieses Konzepts ist Zeit- und Arbeitsersparnis für den Benutzer. Jede existierende Zeichnung kann dabei als Prototyp herangezogen werden. Prototypen beinhalten jegliche Informationen, die man in eine Zeichung einbringen kann. So können z.B. reine Zeichenobjekte, wie Linien oder Rechtecke, gespeichert oder auch Zeichensätze und Linientypen festgelegt werden. Menüs und Dialogfenster lassen sich in gleicher Art und Weise an eine bestimmte Zeichnung binden. Wird eine Zeichung als Prototyp für eine neue Zeichnung zugrunde gelegt, dann werden alle Informationen des Prototyps in die neue Zeichnung übernommen, d.h. diese Daten brauchen nicht erneut erzeugt zu werden. Menüs, die an diesen Prototypen gebunden wurden, stehen dem Benutzer dann automatisch zur Verfügung. Die Erstellung eines Prototyps für einen bestimmten Typ von Zeichnung ist in der Praxis sinnvoll und sehr effektiv, um dem Konstrukteur eine Arbeitsgrundlage zu liefern, von der aus mit der eigentlichen Konstruktion begonnen werden kann.

### Layer

Der Benutzer kann Teile seiner Zeichnung in verschiedenen *Layern* plazieren. Ein Layer wird als eine transparente Zeichnung gesehen. Eine Zeichnung kann so aus einer Menge von Layern, die sich jeweils einzelnd ein- und ausschalten lassen, bestehen indem diese *übereinander* gelegt werden. In der Praxis wird jedem Layer ein bestimmter Aspekt einer Zeichnung zugeordnet. Beispielsweise könnte ein Grundriß eines Hauses auf einem Layer gezeichnet werden und die elektrischen Leitungen auf einem anderen. Jeder einzelne Layer bechreibt somit einen bestimmten Aspekt der Zeichnung, beide zusammen ergeben das Gesamtobjekt.

### PostScript

Mit der PostScript-Unterstützung ergibt sich eine bessere Kompatiblität zwischen AutoCAD, anderen CAD-Systemen und Zeichentools. Folgende Punkte stehen dabei im Vordergrund:

- Benutzung von PostScript-Zeichensätzen in den AutoCAD-Zeichnungen

- Exportieren von Zeichnungen als *Encapsulated PostScript (EPS) Files*
- Importieren von PostScript-Files
- Ergänzung der AutoCAD-Zeichensätze um neue Zeichen
- Definition eigener Füllmuster für Flächen

### AutoLISP

AutoLISP [Autodesk, 1992e], eine Implementation der Programmiersprache LISP [Mayer, 1995], ist Teil des AutoCAD-Pakets. AutoLISP ermöglicht dem Benutzer bzw. dem Applikationsentwickler, Programme und Funktionen in einer höheren Programmiersprache zu schreiben, und so eine CAD-Applikation, zu der diese Funktionen zusammengefaßt werden, entsprechend seinen Bedürfnissen zu entwickeln. Eine detaillierte Beschreibung dieser Möglichkeiten erfolgt in Abschnitt 3.3.2.

### AutoCAD Development System (ADS)

Die Programmierschnittstelle *AutoCAD Development System (ADS)* [Autodesk, 1992a] erlaubt es, eine der Programmiersrpache C [Kernighan und Ritchie, 1990] ähnliche Sprache zur Entwicklung von AutoCAD-Applikationen einzusetzen. Existierende Applikationen bzw. Programme können so genuzt oder eigene programmiert werden. Die ADS-Schnittstelle stellt eine Reihe von C-Bibliotheken zur Verfügung, deren Funktionalität durch äquivalente Funktionen auch auf der Ebene der Programmiersprache AutoLISP verfügbar ist. ADS-Applikationen sind nicht als autonome „Stand-Alone“-Applikationen zu sehen, sondern als Zusammenfassung von externen C-Subroutinen, die vom AutoLISP-Interpreter geladen und gestartet werden (siehe Abschnitt 3.3.2).

### Advanced Modelling Extension (AME)

Das Paket AME [Autodesk, 1992b] ermöglicht dem Anwender eine Modellierung von 2D- bzw. 3D-Objekten. Dieses Paket wird von AutoCAD als externe Applikation zur Verfügung gestellt und kann über die ADS-Schnittstelle geladen und benutzt werden. Dieses Paket ist zur Darstellung und Verwaltung, d.h. also zur Modellierung von realen Objekten grundlegend und wird in Abschnitt 3.2 genauer betrachtet.

### AutoCAD SQL Extension (ASE)

Das Paket *AutoCAD SQL* [Autodesk, 1992f] ist eine mengenorientierte Anfrageschnittstelle zwischen AutoCAD und Datenbank-Management-Systemen (DBMS). Graphikobjekte können mit diesem Paket in externen Datenbanken, wie dBASEIII+, dBASEIV, ORACLE und INFORMIX, verwaltet werden. ASE beinhaltet zusätzlich ein C-Interface, ASI, mit dem eigene SQL-Applikationen erstellt werden können, die von AutoCAD aus mit externen Datenbanken kommunizieren.

**Entities und erweiterbare Entitydaten**

*Entity* ist die in AutoCAD verwendete Bezeichnung für mögliche graphische Elemente, wie Linien, Kreise, Punkte, etc. Mit Hilfe von AutoLISP oder ADS-Programmen ist es bei jedem dieser Entities möglich, die schon durch AutoCAD existierenden Daten, wie z.B. der Radius eines Kreises, um semantische, benutzerspezifische Daten zu erweitern. Diese Daten können einfache Textstrings oder numerische Werte sein, oder auch komplexere Strukturen, wie z.B. 3D-Punkte. So können zu jedem Objekt zusätzliche Informationen gespeichert werden, die über rein geometrische Daten hinausgehen.

## 3.2 Advanced Modelling Extension (AME)

AutoCAD stellt sowohl ein mächtiges *Zeichentool* als auch ein sehr effektives Werkzeug zum Modellieren von Objekten dar. Der Begriff „Zeichentool" soll andeuten, daß AutoCAD dazu genutzt wird, mit Hilfe der zur Verfügung stehenden Graphikelementen, wie Punkt, Linie, Kreis, Rechteck usw. eine Zeichnung zu erstellen. AutoCAD unterstützt den Anwender dabei mit hilfreichen Funktionen, wie z.B. Kopieren, Löschen, Verschieben von Graphikelementen oder Verbinden zweier Linien durch eine Krümmung. Der Aspekt, der ein CAD-System aber erst ausmacht, läßt sich aber nicht durch ein einfaches Zeichentool berücksichtigen. Ein CAD-System muß Werkzeuge bereitstellen mit denen *geometrisches Modellieren* von Objekten möglich ist. Was hinter diesem Begriff steht wurde in Kapitel 2 erläutert.

Das AutoCAD-Paket, das das geometrische Modellieren unterstützt, heißt *Advanced Modelling Extension (AME)* [Autodesk, 1992b]. Als externe Applikation, d.h. als C-Bibliothek implementiert, kann es über die ADS-Schnittstelle geladen und sowohl von C als auch von AutoLISP benutzt werden. Es stellt die Software für 2D- als auch für 3D-Objekte dar. Geschlossene zweidimensionale Objekte, d.h. Flächen, heißen in diesem Kontext *Regions*, und geschlossene dreidimensionale Objekte heißen *Solids*. Die Vorstellung des Modellkonzeptes in AutoCAD erfolgt in Abschnitt 3.2.1. Eng verbunden mit der Entscheidung für ein bestimmtes Modellkonzept ist das Ziel, eine einfache und effektive Handhabung dieses Modells zu erreichen. Die Möglichkeiten, die das ausgewählte Modellkonzept bietet, müssen also für den Benutzer leicht zugänglich und handhabbar sein. Zeichenobjekte, sowohl 2D als auch 3D, müssen sich leicht erzeugen, verändern und erweitern lassen. Verschiedene Perspektiven auf ein Objekt sollten möglich sein. Diese Ziele werden durch das Konzept der *Koordinatensysteme* errreicht. Die AutoCAD Koordinatensysteme werden in Abschnitt 3.2.2 betrachtet.

### 3.2.1 Modellkonzept von AutoCAD

In Kapitel 2 wurden verschiedene Konzepte zur Modellierung von 2D- bzw. 3D-Objekten vorgestellt. Dies waren das *Drahtmodell*, das *Flächenmodell* und das *Volumenmodell*. AutoCAD bedient sich zur Darstellung und Modellierung von Objekten einer Kombination aller drei Konzepte. Eine Zusammenfassung der drei Konzepte wird durch die Implementierung des in Abschnitt 2.2.3 vorgestellten CSG-Verfahren errreicht. Dabei wird prinzipiell nicht zwischen Regions und Solids unterschieden, d.h. sowohl die Modellierung von 2D- als auch 3D-Objekten wird mit dem CSG-Verfahren erreicht, obwohl das CSG-Verfahren ursprünglich für volumenorientierte Darstellungen, also 3D-Darstellungen, gedacht war.

### Constructive Solid Geometry (CSG)

Das CSG-Verfahren (siehe Abschnitt 2.2.3 und 2.3.2) baut auf *primitiven Solids* bzw. *primitiven Komponenten* auf. AME kennt sechs Kommandos, um solche einfachen Solids zu erstellen (siehe Abbildung 3.1):

- `SOLBOX` (Box oder Cube)
- `SOLWEDGE` (Wedge)
- `SOLCONE` (Cone)
- `SOLCYL` (Cylinder)
- `SOLPHERE` (Sphere)
- `SOLTORUS` (Torus)

Abbildung 3.1 zeigt in entsprechender Reihenfolge von links nach rechts beispielhafte primitive Komponenten, die mit diesen Kommandos entstanden sind. Diese primitiven Komponenten dienen dann als Grundlage für komplexere Objekte.

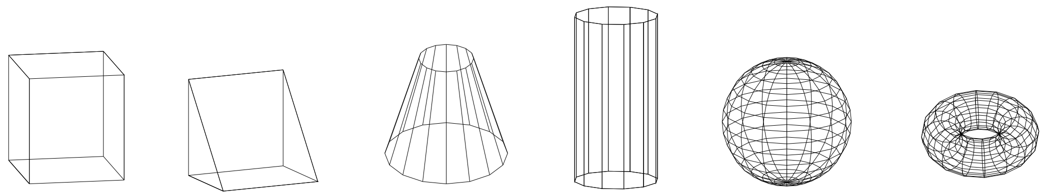

Abbildung 3.1: Primitive 3D-Objekte

Zusätzliche *Solids* können mit AutoCAD durch die Kommandos `SOLEXT` bzw. `SOLREV` erzeugt werden. Diese Kommandos werden auf ein 2D-Entity angewendet und erzeugen durch Verschiebung bzw. Rotation entlang einer Leitlinie einen neuen Volumenkörper, der wiederum als primitive Komponente verwendet werden kann. Diese Technik wurde schon in Kapitel 2.3.2 vorgestellt. AME kennt dagegen keine Kommandos zur Erzeugung primitiver Regions. Diese ergeben sich durch Anwendung des Kommandos `SOLIDIFY` auf beliebige geschlossene 2D-Entities. Als 2D-Entities dienen hier z.B. Kreise, Rechtecke oder 2D-Polylinien.

Werden mehrere Solids oder Regions kombiniert, um ein neues Objekt zu generieren, ergibt sich eine *komplexe Komponente*. Diese komplexe Komponente besteht somit aus einer Menge von Solids oder Regions, aber nicht aus beiden gleichzeitig. Eine komplexe Komponente besteht aus Primitiven, anderen komplexen Komponenten oder aus einer Mischung von beiden. Das CSG-Verfahren erfordert bei der Erzeugung einer komplexen Komponente die Angabe einer Kombinationsvorschrift, wie die entsprechenden Komponenten zu einem neuen Objekt zusammengefaßt werden sollen.

AME hat sieben verschiedene solcher Kombinationsvorschriften implementiert:

- `SOLUNION` (Vereinigung)
- `SOLINT` (Schnitt)

- `SOLSUB` (Subtraktion)
- `SOLREV`
- `SOLEXT`
- `SOLCHAM`
- `SOLFILL`

Die Kommandos `SOLREV` bzw. `SOLEXT` erzeugen, ähnlich wie im obigen Abschnitt, ein neues 3D-Objekt, doch als Grundlage dient hier eine Region, die *Löcher* beinhaltet. `SOLCHAM` bzw. `SOLFILL` ermöglichen es, daß Ecken von Regions oder Solids abgeschrägt bzw. abgerundet werden.

#### Boundary Representation

Die CSG-Technik ist sehr effektiv zur Definition von komplexen Objekten, sowohl 2D als auch 3D. Zusätzlich benutzt das AME aber ein weiteres Verfahren zur Modellierung von Objekten: die *Boundary Repräsentation (B-rep)* [Autodesk, 1992b; Zeid, 1991]. Diese dient hauptsächlich der Darstellung des Objekts auf dem Bildschirm. Somit werden bei jedem Objekt CSG-Informationen über die Struktur und Aufbau des Objekts und B-rep-Informationen zur Beschreibung der Umrandungen des Objekts verwaltet.

Das B-rep-Konzept ist eigentlich nichts anderes als eine Kombination von Draht- und Flächenmodell, denn die Informationen, die hier verwaltet werden, sind Beschreibungen der Umrandungen des Objekts, also Punkte, Kanten und Flächen. Um die Daten aus Drahtmodell und Flächenmodell zu trennen, wird das darzustellende Objekt in diesen beiden Modellvarianten auf zwei verschiedenen Layern (siehe Abschnitt 3.1) dargestellt.

Die Verwendung von CSG- und B-rep-Verfahren erhöht zwar den Verwaltungs- und Berechnungsaufwand, erlaubt aber im Gegenzug eine bessere Genauigkeit sowohl der Visualisierung als auch der Objektanalyse. Durch das Drahtmodell ergibt sich eine genauere Darstellung des Objekts, da sich das Objekt aus Linien, Kreisen, Kreisbögen, etc. zusammensetzt, wo hingegen zur Darstellung des Objekts als Flächenmodell nur Linien benutzt werden und das Objekt durch Flächen approximiert wird. Der Nachteil der sich so ergebenden Formuntreue wird durch den Vorteil ausgeglichen, die vorderen sichtbaren Flächen schraffieren und dahinterliegende Linien ausblenden zu können (siehe Kapitel 2).

### 3.2.2 Koordinatensysteme

AutoCAD benutzt *kartesische Koordinatensysteme*, um Punkte in einer Zeichnung zu lokalisieren. Die X-Achse stellt die horizontale Bemessungen der Zeichnung und die Y-Achse die vertikale Bemessung dar. In diesem Koordinatensystem wird ein 2D-Punkt durch ein $(x, y)$-Koordinatenpaar ausgedrückt. Wenn die Zeichnung einen 3D-Aspekt vorraussetzt, dann wird das Tupel um einen dritten Parameter erweitert, so daß sich für 3D-Punkte ein $(x, y, z)$-Koordinatentripel ergibt. Der $z$-Parameter spezifiziert einen Wert auf der $Z$-Achse, die vertikal auf der $xy$-Ebene steht. Die $xy$-Ebene wird auch als *Konstruktionebene* bezeichnet. Zur Lokalisierung eines Punktes können auch *Polarkoordinaten*, d.h. Winkelangaben, verwendet werden. Diese werden dann vom System in das kartesische Koordinatensystem transformiert.

### World Coordinate System (WCS)

Das Koordinatensystem, das als globale Koordinationshilfe für jede AutoCAD-Zeichnung dient, heißt *World Coordinate System (WCS)*. Es ist fest und unveränderlich. Für 2D-Zeichnungen ist dieses Koordinatensystem ausreichend.

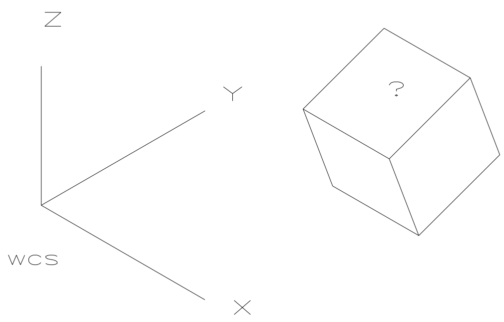


Abbildung 3.2: Probleme mit dem WCS

### User Coordinate Systems (UCS)

Zusätzlich zum WCS besteht die Möglichkeit eigene, d.h. benutzerspezifische Koordinatensysteme (*User Coordinate Systems, UCS*) anzulegen. Der Ursprung kann beliebig in der Zeichnung bzw. in dem zu modellierenden Objekt lokalisiert werden. Die Axen dieses Systems können entsprechend einem gewünschten Blickwinkel auf ein Objekt gedreht werden.

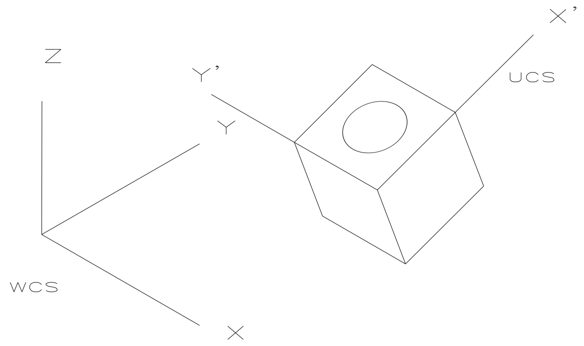


Abbildung 3.3: Benutzerkoordinatensystem

Das in Abbildung 3.2 skizzierte geometrische Problem zeigt, daß das Konzept des USC bei der Modellierung von 3D-Objekten sinnvoll und effektiv ist. Auf der Top-Seite des dargestellten Quaders, der bezüglich der Konstruktionsebene, d.h. $xy$-Ebene, rotiert ist, soll ein Kreis gezeichnet werden. Benutzt man sofort das CIRCLE-Kommando, so liefert dies einen Kreis, der in der $xy$-Ebene erscheint und somit nicht auf der Top-Seite liegt, da die Seite des Quaders und die $xy$-Ebene nicht parallel zueinander sind. Die Lösung dieses Problems zeigt Abbildung 3.3. Mit Hilfe des **UCS**-Kommandos wird ein neues Koordiantensystem definiert, dessen Ursprung an die Ecke des Quaders und dessen $X'$- und $Y'$-Achsen an die Seiten der oberen Quaderfläche gebunden werden können. Wird jetzt ein Kreis gezeichnet so liegt dieser in der $x'y'$-Ebene des neuen UCS und somit auf der Top-Seite des Quaders.

Innerhalb des WCS können beliebig viele UCS definiert sein, wobei aber zu einer Zeit nur ein Koordinatensystem aktiv sein kann. Zu diesem Koodinatensystem sind dann alle weiteren Koordinateneingaben relativ.

## 3.3 Benutzerschnittstellen von AutoCAD

AutoCAD stellt dem Anwender zwei Arten von Benutzerschnittstellen zur Verfügung. Die eine Schnittstelle ermöglicht eine Benutzung des AutoCAD-Systems über ein Text- bzw. Graphikfenster (siehe Abschnitt 3.3.1). Die zweite Schnittstelle erlaubt es, eigene problemspezifische CAD-Applikationen auf Basis des AutoCAD-Systems zu erstellen. Abschnitt 3.3.2 befaßt sich mit den dafür von AutoCAD bereitgestellten Programmiersprachen AutoLISP und C.

### 3.3.1 AutoCAD-Fenster

Die hier verwendete *AutoCAD Version 12* ist eine SPARC-Version und OpenWindows-kompatibel. Die Bedienung der Oberfläche sollte den Benutzern von OpenWindows geläufig sein und wird hier nicht weiter beleuchtet. Aber auch für andere Systeme gibt es eine entsprechende AutoCAD-Implementierung.

AutoCAD stellt als Benutzerschnittstelle ein Text- und eine Graphikfenster zur Verfügung. Auf Zweibildschirmsystemen lassen diese sich auch physikalisch trennen. Befehlseingaben können wahlweise über eine Kommandozeile im Text- bzw. Graphikfenster mit Hilfe der Tastatur oder über verschiedene Menüs mit der Maus erfolgen. Diese Art der Systembenutzung heißt *Dialogbetrieb*. Die Verarbeitung dieser Kommandosprache erfolgt interpretierend, d.h. die Eingabe wird entschlüsselt, auf Korrektheit überprüft und direkt in Unterprogrammaufrufe umgesetzt, wobei gegebenenfalls noch Parameter aus der Eingabe extrahiert und mit an das Unterprogramm gegeben werden.

#### Textfenster

Abbildung 3.4 zeigt ein Textfenster, wie es von AutoCAD beim Start geöffnet wird. Das Textfenster beinhaltet eine Komandozeile, die zur Befehlseingabe verwendet wird. Zusätzlich dient es sowohl zur Ausgabe von Fehler- und Ablaufmeldungen als auch zur Eingabe von Befehlsparametern.

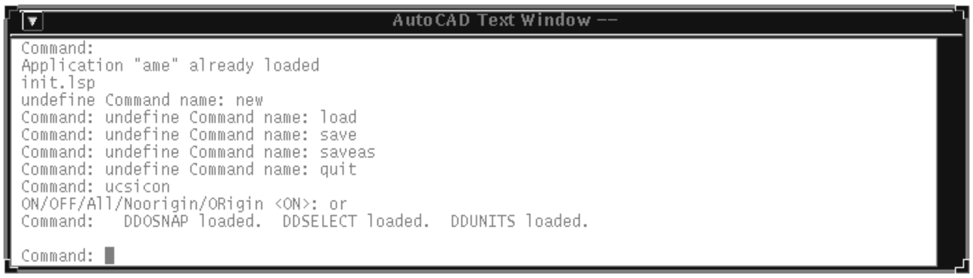


Abbildung 3.4: AutoCAD-Textfenster

### Graphikfenster

Das Graphikfenster von AutoCAD ist beispielhaft in Abbildung 3.5 dargestellt. Hauptelement des Fenster ist die Zeichenfläche. Diese ist umgeben von einer Menüleiste an der oberen Seite, einem Bildschirmmenü an der rechten Seite und einer Kommandozeile an der unteren Seite. Die Kommandozeile des Graphikfensters ist identisch zu der im Textfenster, zeigt dabei aber nur einen kleinen Ausschnitt aus dem Textfenster.

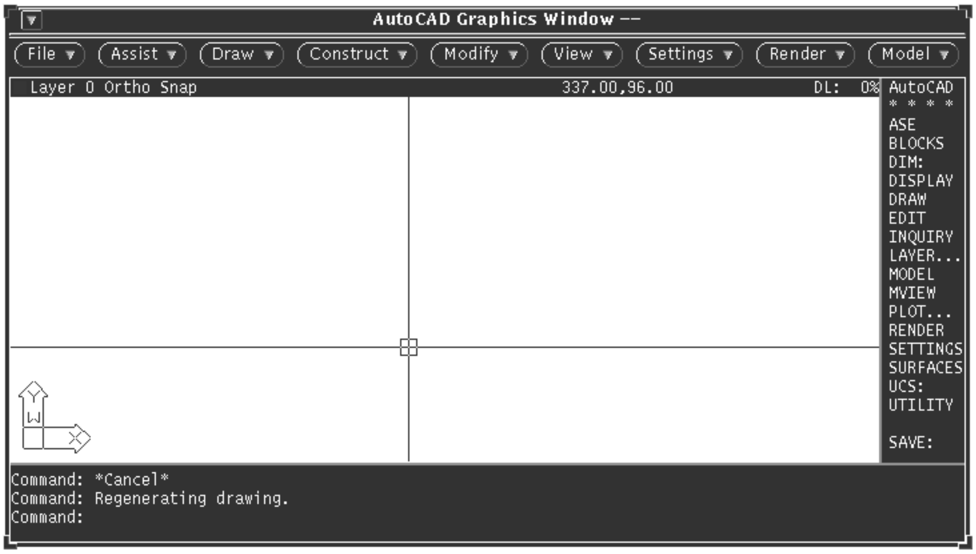


Abbildung 3.5: AutoCAD-Graphikfenster

Folgende verschiedene Typen von Menüs existieren in AutoCAD, die, wie schon erwähnt, alle auf spezifische Benutzerwünsche hin angepaßt werden können (siehe Abschnitt 3.1):

**Bildschirmmenü:** Dieses Menü kann Kommandonamen beinhalten, die durch die Selektion mit der Maus aktiviert werden können. Die Kommandos werden in Form einer Liste an der rechten Seite des Graphikfensters dargestellt (siehe Abbildung 3.5). Mit Hilfe des Bildschirmmenüs kann der Arbeitsfluß des Benutzers logisch geordnet werden.

**Abrollmenüs:** Diese Menüs lassen sich durch Selektion der Knöpfe in der Menüleiste des Graphikfensters *aufklappen*. Da von diesen Menüs neue Abrollmenüs aktiviert werden können, ergibt sich durch diesen Mechanismus eine Kaskadierung bzw. ein hierarchischer Aufbau dieser Menüs.

**Cursormenü:** Dieser Konzept ist ähnlich dem der Abrollmenüs. Das Cursormenü wird aber an der Stelle des Cursors innerhalb der Zeichenfläche aktiviert. Sinnvoll ist dieses Konzept, wenn duch das Cursormenü Kommandos aufgerufen werden, die sehr häufig verwendet werden, wie z.B. der *Object Snap* (siehe Abschnitt 4.4.1).

## 3.3.2 Applikationsentwicklung mit AutoCAD

Die meisten CAD-Systeme bieten dem Benutzer eine breitgefächertes Angebot an geometrischen Funktionen. Viele CAD-Probleme lassen sich mit den bereitgestellten Funktionen lösen. Für einige problemspezifische Aufgaben steht aber keine ausreichende Funktionalität zur Verfügung, zumindest nicht in einer kompakten und benutzerfreundlichen Form. Um diesen Mangel zu beheben, beinhalten einige CAD-Systeme Schnittstellen, die über die Programmiersprachen zugänglich sind, die dem Benutzer die Anpassung des jeweiligen CAD-Systems an die Anforderungen des Einsatzbereiches ermöglichen. Diese Überlegungen entsprechen dem Konzept der *offenen Architektur*, auf dem die Entwicklung von AutoCAD basiert (siehe Abschnitt 3.1). Aus diesem Grund stellt AutoCAD dem Benutzer die eigenentwickelte Interpretersprache *AutoLISP* zur Verfügung. Zusätzlich können über die *ADS-Schnittstelle* (siehe Abschnitt 3.1) C-programmierte Applikationen erstellt und verwendet werden. Eine detailliertere Beschreibung von AutoLISP und C findet sich in den nächsten zwei Abschnitten.

### AutoLISP

AutoLISP [Autodesk, 1992e] ist eine Implementation der Programmiersprache LISP zur Unterstützung der Applikationserstellung innerhalb von AutoCAD. LISP ist eine Sprache, die viele Dialekte kennt, wie z.B. MacLISP, InterLISP, oder CommonLISP [Mayer, 1995]. AutoLISP orientiert sich stärker an der Syntax und den Konventionen von CommonLISP, allerdings mit eingeschränkter Funktionalität und entsprechend den Bedürfnissen eines CAD-Systems erweitert.

Der Entscheidung, LISP als Programmiersprache unter AutoCAD zur Verfügung zu stellen, liegen folgende Argumente zugrunde [Autodesk, 1992e]:

- LISP ist unter allen Programmiersprachen am leichtesten zu erlernen.
- Es ist lange Zeit die ausgewählte Sprache für Entwicklung von KI-Systeme gewesen.
- Aufgrund der einfachen Datentypen Liste und Atom ergibt sich ein ziemlich einfacher Interpreter und ein geringer Speicherbedarf.
- Aufgrund des Interpreter-Konzepts ist ein Testen von Programmcode direkt in der Kommandozeile möglich, ohne Compilieren zu müssen.

Die Funktionalität von AutoLISP umfaßt folgende Bereiche:[1]

- Objektverarbeitung (AutoLISP)
  - Abfragen der Definitionsdaten eines Entities
  - Modifizieren der Definitionsdaten eines Entities
  - Erstellen eines neuen Entity und Hinzufügen zur Zeichnung
  - Löschen von Entities in der Zeichnung oder Zurückholen in die Zeichnung
  - Aktualisieren der Bildschirmanzeige eines Entities
- geometrische Aufgaben (AutoLISP)
- Benutzereingaben (AutoLISP)
- Konversion von Datentypen (AutoLISP/LISP)
- Koordinationstransformation (AutoLISP)

[1] In Klammern ist jeweils angegeben, ob dieser Aufgabenbereich sich nur auf LISP bezieht, oder sich durch die Erweiterung zu AutoLISP ergibt.

- Anzeigesteuerung (AutoLISP/LISP)
- Auswahlsätze (AutoLISP)
- arithmetische Aufgaben (LISP)
- Symbolverarbeitung (LISP)
- Manipulation von Zeichenketten (LISP)
- Kontrollstrukturen (LISP)
- Listenverarbeitung (LISP)
- Verarbeitung von Funktionen (LISP)
- Speicherverwaltung (LISP)
- Verwaltung von Dateien (LISP)
- Fehlerbehandlung (AutoLISP)
- Benutzung von externen ADS-Applikationen (AutoLISP)
- Programmierung von Dialogfenstern (AutoLISP)

AutoLISP unterstützt folgende Datentypen:

**Symbole:** Symbole sind synonyme Variablen und dienen der Speicherung von Werten. Dabei können sie vier Datentypen haben:

- Ganzzahlwert (Integer)
- Realwert (Real)
- Punkt
- Zeichenkette (String)

**Listen:** Die Liste ist eines der wichtigsten Datentypen in LISP. Sie erlaubt mehrere zusammenhängende Werte unter einem Symbol zu speichern. So werden z.B. 2D- und 3D-Punkte als Listen mit zwei bzw. drei Gleikommanzahlen ausgedrückt.

**Dateideskriptoren:** Mit Hilfe dieser Descriptoren werden geöffnete Dateien zum Lesen oder Schreiben referenziert.

**Entitynamen:** Ein Entityname ist ein Zeiger auf ein Entity in der Zeichnung und ist als numerische, von AutoCAD vergebene Kennung implementiert. Diese Kennung erlaubt es, die internen Daten eines Entities zu referenzieren (z.B. <Entity name: 60000016>).

**Auswahlsätze (Selection Sets):** Auswahlsätze sind Gruppen von ein oder mehreren Entities. AutoLISP-Funktionen zur Verwaltung dieser Auswahlsätze ermöglichen z.B. das Generieren von Selection Sets, das Hinzufügen oder das Entfernen von Entities.

**Subroutinen (built-in-Subroutinen):** Mit diesen Funktionen ist die in AutoLISP von Grund auf vorhandene Funktionalität gemeint.

**Externe Subroutinen (externe Subroutinen):** Diese Bezeichnung deutet auf die Funktionalität, die durch externe Applikationen über die ADS-Schnittstelle zur Verfügung gestellt werden kann.

### ADS und C

Das *AutoCAD Development System (ADS)* [Autodesk, 1992a] ist eine auf der Programmiersprache C [Kernighan und Ritchie, 1990] basierende Entwicklungsumgebung für AutoCAD-Applikationen. Obwohl diese Applikationen in C geschrieben sind, werden sie von AutoCAD als AutoLISP-Funktionen betrachtet, da sie über die ADS-Schnittstelle an AutoLISP gebunden werden. Aus diesem Grund stellen ADS-Applikationen keine von AutoCAD unabhängig lauffähigen Programme dar. Sie sind vielmehr als ein Satz von geladenen externen C-Funktionen zu sehen, die vom LISP-Interpreter aufgerufen werden. Auch wenn man sich entscheiden würde, eine AutoCAD-Applikation vollständig in C zu implementieren, geht kein Weg daran vorbei, die Anbindung an AutoCAD durch eine LISP-Routine vorzunehmen.

Die ADS-Entwicklungsumgebung besteht, ähnlich wie in anderen C-Umgebungen, aus einer Funktionsbibliothek und diversen Definitionsdateien (Headerfiles). So ergibt sich eine gute Portabilität des Quellcodes von ADS-Applikationen zwischen verschiedenen AutoCAD-Plattformen. Entwickler benutzen Compiler und Linker der lokalen Systemumgebung. Vorteil einer in C programmierten Applikation ist die gegenüber LISP schnellere Verarbeitungsgeschwindigkeit. Zudem kann Hardware- und Betriebssystemfunktionalität direkt von C aus angesprochen werden. Somit ist die C-Schnittstelle geeignet für Applikationen, die von Rechnerkommunikation Gebrauch machen.

Jede ADS-Applikation benutzt die ADS-Schnittstelle zu AutoLISP. Um externe Funktionen an AutoCAD zu binden, ist eine bestimmte, festgelegte Reihenfolge von Bibliotheksfunktionsaufrufen erforderlich. Folgende Punkte skizzieren die notwendige Aufrufsequenz:

- Initialisierung der Schnittstelle zwischen AutoLISP und der geladenen Applikation
- Anzeige der Bereitschaft zur Abarbeitung von AutoLISP-Anfragen durch die Applikation
- Definition der externen Funktionen der Applikation
- Anzeige der fortdauernden Bereitschaft zur Abarbeitung von AutoLISP-Anfragen

Diese Sequenz läßt erkennen, daß jede ADS-Applikation eigentlich ein „slave" von AutoLISP ist und passiv bleibt bis zur expliziten AutoCAD-Anforderung einer externen Funktionsausführung. Dagegen bleibt bei der Abarbeitung einer externen Funktion AutoLISP bzw. AutoCAD passiv, d.h. es wird auf eine Rückanfrage oder Antwort der ADS-Applikation gewartet. Während dieser Zeit kann AutoCAD auf keine Benutzereingabe reagieren. Diese Art der Kopplung nennt man *synchrone Kommunikation.*

Die Funktionalität der ADS-Bibliotheken ist äquivalent zu der von AutoLISP. Es existieren zwar auch einige Funktionen, die nur in C bzw. AutoLISP verfügbar oder nur dort sinnvoll sind, aber den meisten AutoLISP-Funktionen stehen entsprechende Implementierungen in C gegenüber. Definierte externe C-Funktionen können wie interne oder benutzerdefinierte AutoLISP-Funktionen aufgerufen werden: AutoLISP-Werte und -Variablen können übergeben werden und die externe Funktion kann dem sie aufrufenden AutoLISP-Ausdruck einen Wert zurückliefern. Dazu müssen die grundsätzlich in C existierenden Datentypen, wie Ganzzahlen, Gleitkommazahlen, Zeichen oder Records, durch neue Datentypen erweitert werden, um die gewünschte Funktionalitätsäquivalenz zwischen AutoLISP und C zu erreichen. Die ADS-Entwicklungsumgebung stellt für dieses Ziel die folgenden zwei Datentypen zur Verfügung.

```
struct resbuf{
   struct resbuf *rbnext; /* Zeiger auf nachfolgende Struktur */
   short restype; /* Typ des im Variantenrecord aktuell abgelegten Wertes */
   union ads_u_val resval;
};

union ads_u_val {
   ads_real rreal;
   ads_real rpoint[3];
   short rint;
   char * rstring;
   long rlname[2];
   long rlong;
   struct ads_binary rbinary;
};
```

Die Kombination des Records `resbuf` in Form einer verketteten Liste ermöglicht die dynamische Verwaltung von Daten unterschiedlichen Typs und unterschiedlicher Speichergröße. Die sich so ergebende Listenstruktur wird benutzt, um Elemente (Entities), Symbolltabelleneinträge, AutoLISP-Listen, Auswahlsätze, etc abzubilden. Der im Record `resbuf` enthaltene Variantenrecord `ads_u_val` nimmt dabei die von AutoCAD und ADS verwendeten, oben beschriebenen verschiedenen Datentypen auf.

## 3.4 Möglichkeiten und Grenzen

Jegliche Einzelheiten und Informationen, die die Zeichnung ausmachen, werden in einer Zeichnungsdatei abgespeichert. AutoCAD ermöglicht dem Benutzer wie auch dem Applikationsentwickler schreibenden oder nur lesenden Zugriff auf viele in dieser Zeichnungsdatei enthaltenen Informationen. Auf diese Weise können Objekte in der Zeichnung manipuliert werden, ohne AutoCAD-Kommandos wie `ERASE`, `COPY` oder `MOVE` zu benutzen, d.h. direkter Zugriff auf Beschreibungsdaten eines Objekts ist möglich. Dies ist besonders bei der Erstellung von CAD-Applikationen wichtig, damit von der gewählten Programmiersprache aus auf Informationen von Objekte in der Zeichnung zugegriffen werden kann.

Die Benutzung von AutoLISP oder der ADS-Schnittstelle ermöglicht eine gute und effiziente Entwicklung von CAD-Applikationen auf der Basis von AutoCAD (siehe Abschnitt 3.3.2). Einen sehr einschneidenden Kompromiß muß man aber leider eingehen. AutoCAD-Version 12 stellt sowohl mit AutoLISP als auch mit C keine objektorientierte Programmiersprache zur Verfügung. Besonders im Hinblick auf *geometrisches Modellieren* (siehe Kapitel 2) von Objekten mit Hilfe von CAD-Systemen ist die Objektorientierung ein mächtiges Werkzeug, das erst bei AutoCAD-Version 13 realisiert ist.

Sehr positiv ist die Tatsache, daß einem Applikationsentwickler die Wahl gelassen wird, zwischen den Programmiersprachen C und Lisp in Form von AutoLISP zu entscheiden. Beide Programmiersprachen haben bei der Programmierung ihre Stärken und Schwächen (siehe Abschnitt 3.3.2). Zu dem erhöhten Aufwand, C-Programme erst noch compilieren zu müssen, kommt eine weitere gravierende Schwäche von C, die hier noch einmal betont werden soll. Aufgrund des gewählten Konzeptes der Anbindung von externen C-Applikationen an AutoLISP ergibt sich eine kleine *Client-Server-Umgebung*, in der die C-Applikation die Rolle eines Servers spielt (siehe Abschnitt 3.3.2). Bei Client-Server-Umgebungen ergibt sich oft die Frage, was passiert, wenn der Server bei der Abarbeitung seiner Aufgabe abstürzt oder eine Fehlermeldung liefert. Gegen solche Probleme ist das AutoCAD-System nicht gerüstet. Bei der Programmierung von externen C-Programmen ist es, als Folge von Fehlern in diesem C-Code, sehr oft zu Abstürzen des kompletten AutoCAD-Systems gekommen. Dies unterstreicht noch einmal die Entscheidung für AutoLISP als hauptsächliche Programmiersprache, bei der ein besserer Fehlerbehandlungsmechanismus vorhanden ist.

Das AME-Paket von AutoCAD stellt Funktionen zur Darstellung von 2D- und 3D-Modellen zur Verfügung. Hinter der Implementierung stehen dabei die Konzepte des Kanten-, Flächen- und Volumenmodells. Die Vorteile aus den verschiedenen Konzepte werden dann zur Modellierung von realen Objekten herangezogen (siehe Abschnitt 3.2.1). Flächen werden, auch im Volumenmodell, approximativ beschrieben, was zur Folge hat, daß Freiformflächen nicht beschrieben werden können. Die durch das AME-Paket verfügbaren Benutzerkoordinatensysteme UCS (siehe Abschnitt 3.2.2) sind zur Lösung von geometrischen Aufgaben im 3D-Bereich ein sehr hilfreiches Instrument.

Das Konzept der *open architecture* ist die grundlegende Idee von AutoCAD. Durch dieses Konzept wird AutoCAD zu einem CAD-System, daß den verschiedensten Bedürfnissen von Anwendern angepaßt werden kann. Durch die Möglichkeit der Installation auf verschiedenen Hard- und Softwareplattformen ergibt sich seine große Verbreitung als CAD-System (siehe Abschnitt 3.1).

# Kapitel 4

# AutoCAD-Applikation zur Konstruktion von rotationssymmetrischen Drehteilen

Ziel der vorliegenden Diplomarbeit ist die Konstruktion von rotationssymmetrischen Drehteilen und Bereitstellung von Problemspezifikationen für die Arbeitsplanerstellung bezüglich dieser Objektdomäne. Das System AutoCAD der Firma AutoDesk wurde dabei als CAD-Basissystem ausgewählt. Grundlagen dieser Entscheidung sind die in Kapitel 3 beschriebenen Eigenschaften und Möglichkeiten, die dieses CAD-System dem Anwender und dem Applikationsentwickler bietet.

In diesem Kapitel wird die enstandene Applikation zur Konstruktion von rotationssymmetrischen Drehteilen vorgestellt. Dabei wird auf eine nähere Beschreibung von Implementierungsdetails verzichtet. Vielmehr werden grundlegende Implementierungsentscheidungen und Konzepte beschrieben. Durch diese Beschreibung wird klar, daß die implementierte Applikation aus einer Mischung von Basiskonzepten des AutoCAD-Systems bzw. Verfeinerung und Anpassung dieser Konzepte und zusätzlichen eigenständigen, d.h. von AutoCAD unabhängigen Konzepten besteht. Dazu gehört sowohl die Repräsentation eines Drehteilwerkstücks und dessen Komponenten (siehe Abschnitt 4.3), als auch die Vorstellung der dem Konstrukteur zur Verfügung stehenden Applikations- bzw. Modellierungsschnittstellen (siehe Abschnitt 4.4).

## 4.1 Zielsetzung

Bei der Entwicklung von CAD-Systemen stehen grundsätzlich bestimmte Anforderugen im Vordergrund, die das System als formale Eigenschaften abdecken sollte. In welchem Ausmaß diese Anforderungen realisiert werden ist dabei abhängig von der konkreten Applikation, bestimmt aber die Güte eines CAD-Systems.

Folgende allgemeine Anforderungen sollten bei der Erstellung eines CAD-Systems, im speziellen für 3D-Objekte, betrachtet werden [Zeid, 1991]:

**Korrektheit:** Zu jeder internen Objektrepräsentation existiert ein reales 3D-Objekt.

**Eindeutigkeit:** Zu jeder internen Objektrepräsentation existiert *genau ein* 3D-Objekt.

**Vollständigkeit:** Jegliche Funktionalität, die das System bereitstellt, muß auf jedes modellierbare Objekt in der ausgewählten Domäne anwendbar sein.

**Redundanzfreiheit:** Bei der Festlegung der Datenstrukturen sollten Informationen nicht mehrfach abgebildet werden.

**Flexibilität:** Die interne Repräsentationen der Objekte sollte leicht erweiterbar bzw. modifizierbar sein.

**Spezialisierung:** Die interne Repräsentation sollte auf die zu erwartende Weiterverarbeitung der CAD-Daten zugeschnitten sein.

**Universalität:** Eine allgemeine und universelle Darstellung der Objekte wäre sinnvoll im Hinblick auf nachgelagerte Arbeitsprozesse.

Einige der genannten Punkte stehen untereinander in einem Konflikt, wie z.B. Universalität und Spezialisierung. Verschiedenen Applikationen liegen verschieden starke Ausprägungen dieser Anforderungen zugrunde. Die Kombination dieser Ausprägungen bestimmt so die Effizienz eines CAD-Systems, wobei die Effizienz sich auf Speicherplatz, Zeitverhalten, Resourcenausnutzung, Verhältnis von Aufwand und Ergebnis, einfache Implementierung oder einfache Handhabung bzw. Modellierung beziehen kann.

Bei dieser Diplomarbeit stehen neben den oben genannten Anforderungen noch weitere Aspekte im Vordergrund. Das **äußere Design** der Applikation und die Art und Weise wie **Parametereingaben** gemacht werden, orientiert sich an AutoCAD, damit äußerlich kein *neues* CAD-System entsteht, sondern damit AutoCAD-Benutzer, die schon vorher mit AutoCAD gearbeitet haben, geringe Umstellungsprobleme haben.

Ein weiterer Aspekt bezieht sich auf den **zeitlichen Aufwand**, den man beim Modellieren auf sich nehmen muß. Bei der implementierten Applikation ergibt sich ein enormer Zeitgewinn, indem Parametereingaben über eine Kommandozeile eingegeben werden können, anstatt diese über ein Dialogfenster zu machen. Praktische Erfahrungen haben zudem gezeigt, daß es beim Modellieren von Objekten nicht auf ganz exakte Dimensionsangaben ankommt, sondern daß sich Parameterwerte für Objekte in einem gewissen Toleranzbereich aufhalten können. Der konzeptionelle Aufbau, d.h. die Gestalt eines Objektes tritt hier in den Vordergrund, ohne daß sich dadurch grundsätzlich andere Eingabedaten für den dem CAD-System nachgelagerten Arbeitsprozeß ergeben. Zudem ist nach einer grundsätzlichen Modellierung der Objektstruktur eine spätere Feinmodellierung des Objektes leicht möglich. Aus diesem Grund steht eine weitere Vorgehensweise bei der Modellierung zur Verfügung. Es ist möglich mit Hilfe der Maus die Ausmaße eines Objektes zu spezifizieren. Daraus ergibt sich eine beträchtliche **Verringerung des Zeitaufwandes** und eine **Verbesserung des Modellierungskomforts**.

Diese Aspekte können natürlich bei beliebigen Objektdomänen in Betracht gezogen werden, hier beziehen sie sich aber im speziellen auf die Domäne der rotationssymetrischen Drehteile.

## 4.2 Grundlegende Implementierungsentscheidungen

Im Kapitel 2 wurde eine Reihe von möglichen Konzepten zur Modellierung von realen Objekten vorgestellt. Teile dieser Konzepte sind als Grundlage für das CAD-System AutoCAD ausgewählt worden (siehe Abschnitt 3.2.1). Dieser Abschnitt beschäftigt sich mit der Art und

Weise, in der die Basiskonzepte von AutoCAD zur Modellierung von rotationssymmetrischen Drehkörpern genutzt werden.

### 4.2.1 LISP als Implementierungssprache

Unter den beiden vorhandenen Implementierungssprachen wurde AutoLISP als die hauptsächliche Sprache ausgewählt. Entscheidungsgrundlage dafür bilden die in Abschnitt 3.3.2 genannten Punkte und damit die Verfolgung des Zieles bezüglich eines optimalen Verhältnisses zwischen Aufwand und Ergebnis. Im zeitlichen Rahmen dieser Diplomarbeit hat die Entscheidung für AutoLISP einen rationalen Ursprung. In Bezug auf die Funktionalität von AutoLISP steht C, wie schon erwähnt, LISP in nichts nach. Hier wurde nur ein kleiner Teil der Implementierung, der sich mit dem Aufbau eines Client-Server-Modells befaßt, in C vorgenommen, da die Kommunikationsmechanismen von UNIX nur in C, aber nicht in AutoLISP verfügbar sind.

### 4.2.2 Interne Modellierung

Die hier vorgestellte CAD-Applikation befaßt sich mit dem Modellieren von rotationssymmetrischen Drehteilen. Wie aus den Darstellungen von CAD-Systemen in Abschnitt 2.3 hervorgeht, liegt hier ein typischen Vertreter der $2\frac{1}{2}$D-Systeme vor. Dies macht die interne Repräsentation des Objektes und die aufbauenden Algorithmen weniger komplex, indem Konturzügen bzw. Flächen eine variable oder konstante Raumtiefe zugeordnet wird, und sich so eigentlich eine 2D-Darstellung und Verarbeitung ergibt.

Grundlage für die Modellierung der Objekte bildet das AME-Packet (siehe Abschnitt 3.2). Die dort realisierten Modellkonzepte sind das *Volumenmodell* als CSG-Verfahren und die *Boundary Representation* als Kombination von Draht- und Flächenmodell. Diese Konzepte sind bezüglich einer reinen Darstellung von 2D- bzw. 3D-Objekten sehr effektiv und nützlich. Leider zeigen sie bei der internen Darstellung eines Objektes, besonders bei der Abbildung der Objekteigenschaften, die wichtiger Teil des Modellierungsprozesses ist, starke Schwächen. So mußten zusätzliche Strukturen implementiert werden, die die interne Darstellung der Objekte übernehmen. Das in Abschnitt 3.1 erwähnte Konzept der *erweiterbaren Entitydaten* reicht hier auch nicht aus, da bei diesem Konzept nur eine sehr begrenzte Datenmenge speicherbar ist.

Die implementierte Struktur zur internen Repräsentation von Werkstücken lehnt sich an das CSG-Verfahren. So ergibt sich eine umgedrehte Baumstruktur, die beispielhaft an einem einfachen Drehteil in Abbildung 4.1 skizziert ist.

Wurzel dieses Baums bildet das komplette Werkstück, das in der zweiten Stufe des Baums aus den einzelnen primitiven Komponenten besteht. Ein komplexes Gesamtwerkstück besteht somit aus der Zusammenfassung, d.h. Aneinanderreihung, der Einzelkomponenten. Im Gegensatz zum CSG-Verfahren ist es nicht möglich, daß primitive Komponenten aus weiteren primitiven Komponenten bestehen, sodaß sich dieses Verknüpfungsprinzip nur auf die erste Ebene des Baumes bezieht. Eine tiefere Verzweigung des Baumes ergibt sich erst durch weitere Bearbeitungsschritte, sogenannte *Features*, für die primitiven Komponenten (siehe Abschnitt 4.3). Die *Abrundung*, *Nut* und *Abschrägung* dienen in der Skizze als Beispiel für solche Bearbeitungsschritte. Die betroffenen primitiven Komponenten halten entsprechende Referenzen auf die Features. Die genauere Beschreibung der einzelnen Features erfolgt in Abschnitt 4.3.3.

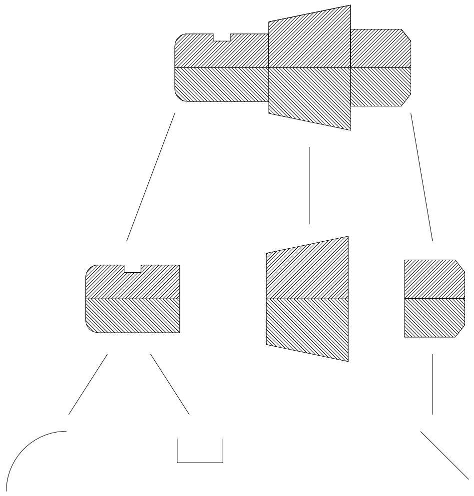

Abbildung 4.1: Interne Baumstruktur eines einfachen Drehteils

Die interne Repräsentation eines Werkstückes durch diese Baumstruktur ist konzeptionell sehr einfach gehalten, da die Implementierungssprache AutoLISP aus Gründen der Einfachheit, Übersichtlichkeit und Geschwindigkeit, es nicht zuläßt, komplexe Datenstrukturen aufzubauen.

### 4.2.3 Darstellung und Identifikation von Objekten

Zur Darstellung von Objekten wird die Funktionalität von *Regions* benutzt, die das AME-Packet von AutoCAD (siehe Abschnitt 3.2) zur Verfügung stellt. Die Randkontur einer *Region* besteht aus Zeichenelementen wie Linien oder Kreisbögen. Aus diesem geschlossenen Kantenzug definiert sich die Region, die dann zu einer Fläche schraffiert wird. Zur Darstellung einer Region steht das Flächen- bzw. das Drahtmodell zur Verfügung. Auf Grundlage des AutoCAD-Flächenmodells wird ein Objekt nur durch Linien approximiert. Die daraus resultierende Formuntreue macht die Benutzung dieses Modells zur Objektdarstellung unmöglich. Zur optischen Repräsentation wird also das Drahtmodell benutzt, das eine exakte Darstellung erlaubt.

Das AutoCAD-System generiert für jegliche Zeichenentities, also auch für Regions, einen eindeutigen Entitynamen (siehe Abschnitt 3.3.2). Über diesen Namen können einzelne Flächen, die ein Objekt ausmachen, identifiziert werden. Über diesen Namen werden dann auch die hinter den einzelnen Flächen stehenden primitiven Komponenten bzw. Eigenschaften der primitiven Komponenten referenziert.

Die Identifikation von einzelnen Komponenten reicht aber noch nicht aus. Zusätzlich müssen auch einzelne Konturelemente referenziert werden können, z.B. wenn der Benutzer bei der Modellierung einer Abschrägung aufgefordert wird, die gewünschte Seite der Komponente (links oder rechts) mit der Maus zu spezifizieren. Das Drahtmodell stellt zwar durch Verwendung von exakten Konturelementen, wie Linie und Kreisbogen, das Objekt sehr genau dar, erlaubt aber keine Identifikation einzelner Konturelemente bzw. nur mit ungerechtfertigt hohem Aufwand. Das Flächenmodell ist zur Identifikation auch ungeeignet, da es nur Linien benutzt. Die Applikation wird also um ein „neues“ Drahtmodell erweitert, daß auf das Problem der Identifikation von Konturelementen zugeschnitten ist. Dieses Drahtmodell läßt die

Identifikation einzelner Konturelemente zu. Die einzelnen Elemente können dann, wie schon erläutert, über den Entityname referenziert werden.

### 4.2.4 Datenstrukturen

Die verschiedenen Objekte, die in die Beschreibung eines Werkstücks eingehen, besitzen sowohl eine Menge von meßbaren Eigenschaften als auch eine Menge von Beziehungen zu anderen Objekten. In diesem Absatz sollen die grundlegenden Datenstrukturen zur Objektrepräsentation dargestellt werden. Hauptaugenmerk wird hier auf Datenstrukturen gelegt, die die in Abbildung 4.1 gezeigte Baumstruktur implementieren. Die Datenstrukturen für die eigentlichen Beschreibungsinformationen der verschiedenen primitiven Komponenten und Features erfolgt erst in Abschnitt 4.3.

Grundlegende Datenstruktur in LISP ist die Listenstruktur. Die Beschreibung sowohl von Objekteigenschaften als auch von Beziehungen zwischen Objekten erfolgt darum in Form von Listen, sogenannten *Assoziationslisten* (siehe [Mayer, 1995]).

Folgende Datenstrukturen sind zur internen Repräsentation der Beziehungsstruktur zwischen dem gesamten Werkstück, primitiven Komponenten und Features implementiert worden:

- `objid`
  Das Synonym `objid` steht für eine Referenz auf eine primitive Komponente. Die Referenz ist dabei ein Identifier der Region, die die entsprechende primitive Komponente graphisch darstellt. Der in Abschnitt 4.2.3 vorgestellte Identifier für ein Zeichenentity ist hier leider unbrauchbar, da er beim erneuten Laden einer Zeichnung vom System neu generiert wird, und so ein Objekt mit dem alten Identifier nicht mehr referenziert werden kann. Parallel zu diesem Identifier generiert AutoCAD einen ähnlichen eindeutigen Identifier, der während der Lebensdauer einer Zeichung bestehen bleibt. Dieser kommt hier zur Anwendung.

- `featid`
  Das Synonym `featid` steht für eine Referenz auf ein Feature. Da ein Feature nicht immer nur aus einem Zeichenentity besteht, wie eine primitive Komponente immer aus genau einer Region besteht, sind die von AutoCAD vergebenen Identifier unbrauchbar. Die Referenzen auf Features sind also selbst generierte Identifier, d.h. nichts anderes als hochgezählte Integerwerte.

- `wp_list`
  Das komplette Werkstück wird nicht durch einen Identifier repräsentiert, sondern durch die `wp_list`. Diese Liste beinhaltet Referenzen bzw. Identifier der primitiven Komponenten, aus denen das Werkstück besteht(siehe `objid`). Die Reihenfolge der Identifier in der Liste entspricht dem Aufbau des Werkstücks von links nach rechts.

- `objid_featid_ass`
  Diese Listenstruktur enthält Informationen darüber, bei welchen primtiven Komponenten welche Features angebracht sind. Primitive Komponenten und Features werden in der Liste mit ihren entsprechenden Identifiern abgebildet (siehe `objid` und `featid`) und nicht mit ihren vollständigen Beschreibungsinformationen.

Abbildung 4.2 zeigt die interne Baumstruktur eines einfaches Drehteils mit den oben genannten benötigten Datenstrukturen und beispielhaften Identifiern für primitive Komponenten und Features. Identifier für primitive Komponenten werden von AutoCAD als Strings

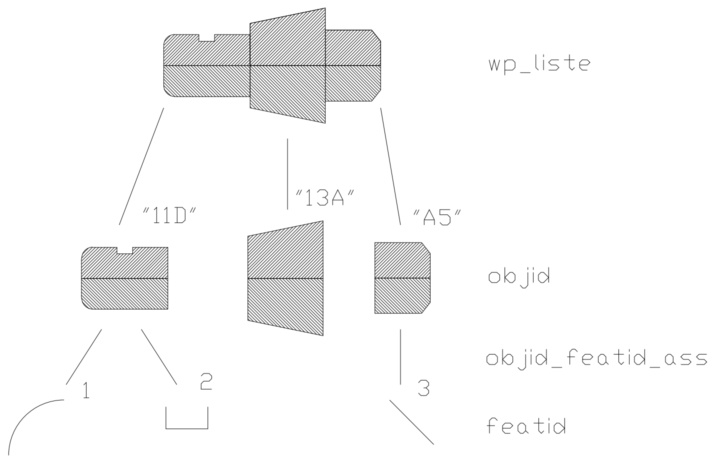


Abbildung 4.2: Beziehungsstrukturen eines einfachen Drehteils

implementiert und die selbst generierten Featureidentifier sind Integerwerte. Mit diesen Beispielwerten ergeben sich für die beiden Listen `wp_list` und `objid_featid_ass` folgende Belegungen.

```
wp_list          = ( "11D" "13A" "A5" )
objid_featid_ass = ( ("11D" (1 2)) ("A5" (3)) )
```

Aus Gründen der Übersichtlichkeit, Fehlen von Objektorientierung, Fehlen eines Datenbrowsers und Beachtung der Implementierungskomplexität ist die Menge der verschiedenen Datenstrukturen und ihre Verknüpfung sehr gering gehalten. Eine Erweiterung ist aufgrund der Einfachheit der vorliegenden Strukturen leicht möglich.

## 4.3 Werkstückrepräsentation

Dieser Abschnitt beschreibt die Komponenten aus denen ein rotationssymmetrisches Drehteil besteht. Hier wird sowohl das äußere Format als auch eine entsprechende Datenstruktur skizziert. Dies macht zusätzlich das Konzept des Modellierungsvorgang von Drehkörpern deutlich.

### 4.3.1 Vorbemerkungen

Die Modellierung eines Drehteils erfolgt prinzipiell durch Aneinanderreihung von rotationssymmetrischen Grundkörpern, den sogenannten *primitiven Komponenten*. Diese werden in Abschnitt 4.3.2 vorgestellt. Die primitiven Komponenten können aber durch sogenannte *Features* noch weiter bearbeitet bzw. verfeinert werden (siehe Abschnitt 4.3.3). Die Features setzen an der Mantelfläche der primitiven Komponenten an.

Jeder Modellierungsprozeß hat stets das Ziel, eine geometrische Beschreibung eines zu konstruierenden Objekts zu liefern. Das Problem dabei ist die automatische Weiterverarbeitung der Konstruktionsdaten in nachgelagerten Arbeitsbereichen, in denen die Geometrie zu interpretieren ist und somit eine ganz bestimmte Bedeutung hat [Roller, 1995]. Dem Konstrukteur geht es bei seiner Arbeit im wesentlich darum, eine festgelegte Funktion zu erfüllen. Für ihn hat die angegebene Geometrie eine eigene und spezielle Bedeutung. Die Interpretation einer erzeugten Werkstückgeometrie, die nur aus einfachen primitiven Komponenten besteht, ist dabei nicht besonders aufwendig und schwierig, da die Vielfalt von primitiven Komponenten sehr begrenzt ist. Schwieriger zu interpretieren sind dabei Verfeinerungen von primitiven Komponenten. Diese geometrischen Teile eines Gesamtmodells werden *Form Features* genannt [Roller, 1995], und werden innerhalb des Produktentwicklungsprozesses als eigenständige Einheit betrachtet. So kann für den Konstrukteur die geometrische Beschreibung in Form einer Spline-Fläche die Abrundung einer Kante darstellen. Typische Beispiele für Form Features sind Nut, Lochkreis, Nabe oder Durchgangsloch. Das Problem der Interpretation von Features erfordert eine genauere Betrachtung.

Der Begriff *Feature* ist nicht streng formal definiert und wird in der Literatur auch nicht ganz einheitlich verwendet. Den meisten Versuchen, diesen Begriff zu klären, liegen folgende Überlegungen zugrunde [Bernadi *et al.*, 1991b]:

- Ein Feature ist eine Abstraktion von Informationen auf niedriger Ebene.
- Featureinformationen sind geometrischer Natur und in den Konstruktionsdaten der Objekte enthalten.
- Features werden einem bestimmten Bezugsobjekt zugeordnet und erhalten erst duch diese Zuordnung ihre Bedeutung.

Einen guten Überblick über die Ansätze zur Definition von Features liefert [Bernadi *et al.*, 1991b].

Bei der Anwendung des Feature-Konzeptes in CAD unterscheidet man zwischen

- Design by Features
- Feature Recognition

Beim *Design by Features* wird es dem Konstrukteur ermöglicht, mit zur Verfügung gestellten Featuremakros den Entwurf von Werkstücken über die Benutzung von primitiven Komponenten hinaus vorzunehmen. Dies führt gleichzeitig zu einer Erleichterung des Konstruktionsprozesses.

Beim Ansatz des *Feature Recognition* geht es darum, Features auf der Grundlage der geometrischen Daten eines Werkstückes automatisch zu erkennen. Dies ist eines der Aufgaben, die von den der Werkstückmodellierung nachgelagerten Arbeitsprozesse zu bewältigen sind. Im Rahmen der Fertigungsplanung ist zu erkennen, welche Bereiche eines Werkstückes aufgrund der Modellierung eines Features einen eigenen Arbeitsschritt erfordern, z.B. eine Bohrung oder eine Abrundung. Die Hautpprobleme bei der Feature Recognition sind Erkennung von Featuretypen und die Verarbeitungsgeschwindigkeit bei der Featureerkennung. Verschiedene Verfahren zur Lösung dieser Probleme sind entwickelt worden [Mauss, 1992; Gossard und Sakurai, 1990].

Bei der vorliegenden Arbeit kann der Benutzer bzw. der Konstrukteur im Verlauf der Konstruktion dem Werkstück bzw. einer primitiven Komponente vordefinierte Features zuordnen. Dies wird so in die interne Werkstückrepräsentation eingebracht, daß nachfolgenden Prozessen, insbesondere der Fertigungsplanung, die Erkennung von Features leicht gemacht wird.

Der aufwendige, unsichere und zeitaufwendige Schritt der Featureerkennung entfällt damit. Features werden nicht *erkannt*, sondern im Rahmen der Konstruktion *modelliert*.

Damit das modellierte Werkstück auch wirklich auf ein reales Werkstück abgebildet werden kann, sind bestimmte *Integritätsbedingungen* beim Erstellen der Objekte einzuhalten. Dies gilt sowohl für primitive Komponenten, als auch für Features. Dabei gibt es Integritätsbedingungen, die offensichtlich und leicht verifizierbar sind, wie die positive Länge eine Objektes. Es gibt aber auch Bedingungen, die beim Modellieren des Werkstücks noch nicht erkennbar oder nicht überprüfbar sind. Die Korrektheit eines Objekts, die zu den Zielen bei der Erstellung eines CAD-Systems zählt (siehe Abschnitt 4.1), läßt sich in diesen Fällen erst bei den nachfolgenden Arbeitsprozessen erreichen. Ein System, das sich mit diesem Problem beschäftigt und dem Modellierungsvorgang nachgelagert wird, ist das System *EMACS* [Nau *et al.*, 1995].

Die Datenstrukturen für primitive Komponenten und für Features werden in den nächsten beiden Abschnitten vorgestellt. Dabei wird aber nur ein Auschnitt der Datenstrukturen behandelt, der für den Leser von Interesse ist. Tiefere Implementierungsdetails werden damit außer acht gelassen.

### 4.3.2 Primitive Komponenten

Zum Aufbau eines Drehteils werden drei Typen von primitiven Komponenten zur Verfügung gestellt, die jeweils eine unterschiedliche Geometrie aufweisen. Die Geometrie einer primitiven Komponente wird durch bestimmte Parameter definiert. Durch Wertebelegung dieser Parameter wird das Ausmaß der Komponente festgelegt.

Bei der Modellierung eines Werkstückes mit der vorliegenden Applikation können Drehteile erstellt werden, bei denen sich die primitiven Komponenten auf zylindrische, konische und tonnenförmige Geometrien beschränken. So sind nicht alle prinzipiell möglichen Drehteile darstellbar. Für die Untersuchung der grundsätzlichen Probleme und Fragestellungen im Bereich der Fertigungsplanung für Drehteile ist dies aber ausreichend.

Die in den nächsten Darstellungen gezeigten Querschnitte der primitiven Komponenten bzw. Features ergeben sich auch so bei der Modellierung der Objekte mit AutoCAD, d.h. es sind 2D-Objekte, obwohl sie Modellierungen von realen dreidimensionalen Objekten mit einem $2\frac{1}{2}$D-System sind. Dies zeigten schon die Entscheidungen aus Abschnitt 4.2. Bei der Modellierung eines Werkstücks wird nur die obere Hälfte dargestellt, da die untere Hälfte äquivalent ist, und sich so eine Vereinfachung für den Benutzer ergibt. Die folgenden Abbildungen zeigen jedoch die vollständigen Querschnitte der primitiven Komponenten.

#### Zylinder

| **Parametername** | **Wert** | **Bedeutung** | **Bedingung** |
|---|---|---|---|
| length | <*Number*> | Länge $L$ | $L > 0$ |
| radius | <*Number*> | Radius $R$ | $R > 0$ |
| left_inner_radius | <*Number*> | Radius $R_l$ der linken Seite des Zylinders | $R_l > 0$ |
| right_inner_radius | <*Number*> | Radius $R_r$ der rechten Seite des Zylinders | $R_r > 0$ |
| obj_type | “cylinder“ | | |

Wenn ein Zylinder Features besitzt, besteht die Möglichkeit, daß der linke bzw. rechte Radius

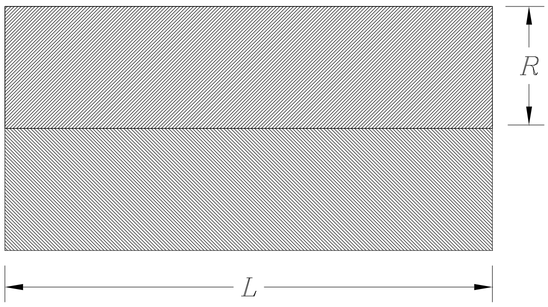


Abbildung 4.3: Zylinderkomponente

nicht mehr mit dem ursprünglichen Grundradius R übereinstimmt. Um die Komponente aber noch ausreichend beschreiben zu können, wird die Zylinderdatenstruktur um die Parameter *left_inner_radius* und *right_inner_radius* ergänzt.

**Kegelstumpf**

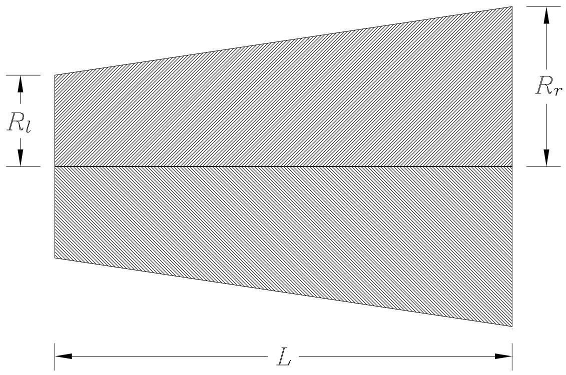


Abbildung 4.4: Kegelstumpfkomponente

| **Parametername** | **Wert** | **Bedeutung** | **Bedingung** |
|---|---|---|---|
| length | <*Number*> | Länge $L$ | $L > 0$ |
| left_radius | <*Number*> | Radius $R_l$ der linken Seite des Kegelstumpfs | $R_l > 0$ |
| right_radius | <*Number*> | Radius $R_r$ der rechten Seite des Kegelstumpfs | $R_r > 0$, $R_l \neq R_r$ |
| obj_type | “cone“ | | |

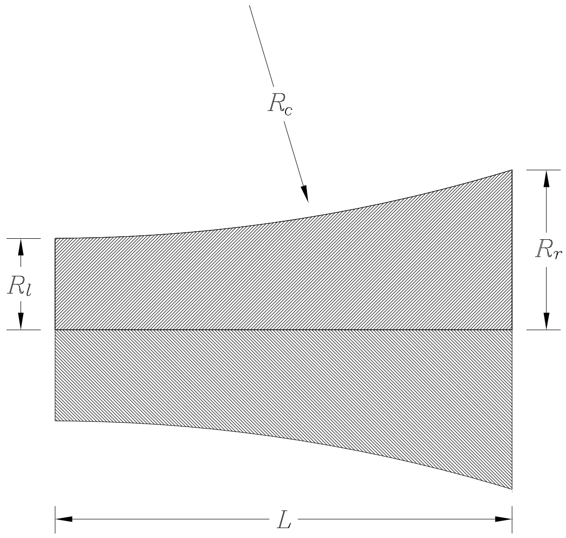


Abbildung 4.5: Tonnenkörperkomponente

**Tonnenkörper**

| Parametername | Wert | Bedeutung | Bedingung |
|---|---|---|---|
| length | <*Number*> | Länge $L$ | $L > 0$ |
| left_radius | <*Number*> | Radius $R_l$ der linken Seite des Tonnenkörpers | $R_l > 0$ |
| right_radius | <*Number*> | Radius $R_r$ der rechten Seite des Tonnenkörpers | $R_r > 0$, $R_l \neq R_r$ |
| c_radius | <*Number*> | Konturradius $R_c$ des Tonnenkörpers | $R_c \geq \frac{\Delta^2+L^2}{2*\Delta}$ mit $\Delta =\mid R_r - R_l \mid$ |
| curvature | {convex, concave} | Krümmung des Tonnenkörpers | |
| obj_type | "toroid" | | |

Die letzte Integritätsbedingung garantiert, daß die Konturlinie eines Tonnenkörpers entweder monoton steigend oder monoton fallend ist. Das System berechnet aus den Parametern *length*, *left_radius* und *right_radius* einen minimal möglichen Konturradius, der dem Benutzer mitgeteilt wird. Die Überprüfung der Einhaltung dieser Bedingung wird vom System übernommen.

Abbildung 4.6 zeigt ein komplettes Werkstück, das aus den oben vorgestellten primitiven Komponenten besteht.

### 4.3.3 Features

Die nächsten Seiten zeigen die Features *Abrundung*, *Abschrägung*, *Gewinde* und *Formeinstich*. Die Vielzahl von denkbaren Featuretypen wird hier also auf die wichtigsten beschränkt. Eine Erweiterung dieser Featuremenge ist leicht möglich, da die grundlegende Struktur und das grundlegende Verhalten anderer Features sich an den hier implementierten Features orien-

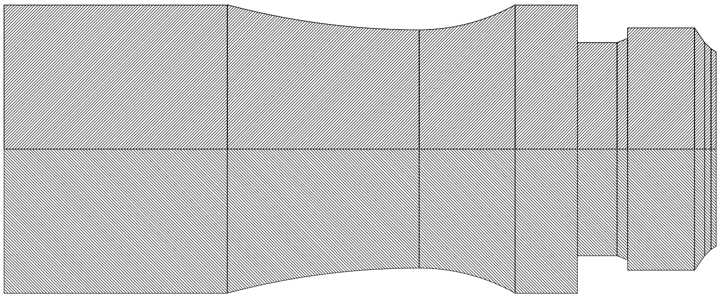

Abbildung 4.6: Beispielhaftes Werkstück

tieren. Die Darstellung von Featuretypen wird in den nachfolgenden Abbildungen an einem Zylinderausschnitt vorgenommen.

### Abrundung

Mit Hilfe einer *Abrundung* kann ein Zylinder an der linken oder rechten Seite abgerundet werden.

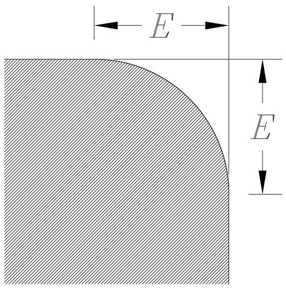


Abbildung 4.7: Abrundung

| **Parametername** | **Wert** | **Bedeutung** | **Bedingung** |
|---|---|---|---|
| extent | $<$*Number*$>$ | Ausdehnung $E$ | $E > 0$ |
| side | {left, right} | Seite des Zylinders, die abgerundet werden soll | |
| obj_type | "round_off" | | |

### Abschrägung

Mit Hilfe einer *Abschrägung* kann ein Zylinder an der linken oder rechten Seite abgeschrägt werden.

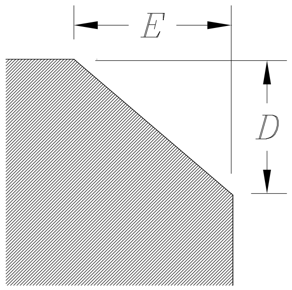


Abbildung 4.8: Abschrägung

| **Parametername** | **Wert** | **Bedeutung** | **Bedingung** |
|---|---|---|---|
| extent | <*Number*> | Ausdehnung $E$ | $E > 0$ |
| depth | <*Number*> | Tiefe $D$ | $D > 0$ |
| side | {left, right} | Seite des Zylinders, die abgeschrägt werden soll | |
| obj_type | “slope“ | | |

**Gewinde**

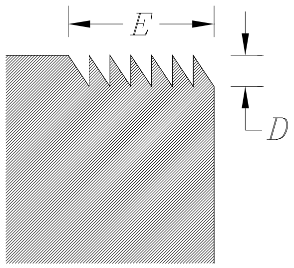


Abbildung 4.9: Gewinde

| **Parametername** | **Wert** | **Bedeutung** | **Bedingung** |
|---|---|---|---|
| extent | <*Number*> | Ausdehnung $E$ | $E > 0$ |
| depth | <*Number*> | Tiefe $D$ | $D > 0$ |
| extent_of_1 | <*Number*> | horizontaler Abstand zwischen zwei benachbarten Zacken | extent_of_1> 0 |
| side | {left, right} | Seite des Zylinders, an der das Gewinde angebracht wird | |
| obj_type | “threat“ | | |

Der Parameter `extent_of_1` ist in Abbildung 4.9 nicht eingezeichnet.

### Formeinstich

Ein Formeinstich wird erzeugt, indem man einen entsprechend geformten Einstichmeißel an das rotierende Drehteil heranführt und so eine Vertiefung ausdreht. Den möglichen Formen dieser Vertiefungen sind prinzipiell keine Grenzen gesetzt (siehe Abbildung 4.10). Die Formeinstiche werden mit einem speziellen Werkzeug, dem *Nuteneditor*, modelliert. Dieser wird in Abschnitt 4.4.2 genauer beschrieben.

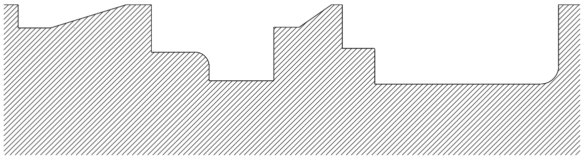

Abbildung 4.10: Beispiele für Formeinstiche

Ein Spezialfall eines Formeinstiches stellt die sogenannte *Nut* dar. Sie entsteht, indem der Einstichmeißel aus dem rotierende Werkstück eine rechteckige Vertiefung ausschneidet (siehe Abbildung 4.11). Da der Featuertyp Nut ein Spezialfall eines Formeinstiches ist, wird nur eine interne Repräsentation zur Verfügung gestellt, die aus der Tiefe und Breite der Nut besteht.

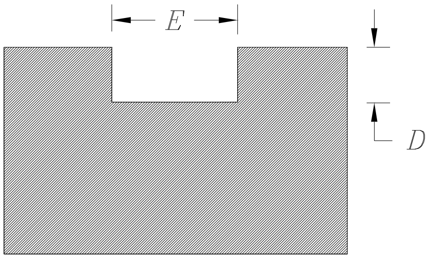


Abbildung 4.11: Nut

| **Parametername** | **Wert** | **Bedeutung** | **Bedingung** |
|---|---|---|---|
| extent | <*Number*> | Ausdehnung $E$ | $E > 0$ |
| depth | <*Number*> | Tiefe $D$ | $D > 0$ |
| obj_type | "groove" | | |

Wie schon unter Abschnitt 4.3.1 erwähnt, müssen auch für Features bestimmte Integritätsbedingungen eingehalten werden, um eine korrekte Abbildung von einem realen Werkstück in ein Modell zu gewährleisten. Hier wird unterschieden zwischen leicht und weniger leicht zu verifizierenden Bedingungen.

Abbildung 4.12 zeigt die Features Abrundung, Abschrägung und Gewinde, wobei eine leichte Überprüfung der Gültigkeit nur bei der Abrundung, also Radius größer Null, möglich ist. Die Überprüfung der Gültigkeit von Abschrägung und Gewinde in bezug auf die korrekte Plazierung wird durch nachfolgende Bereiche übernommen.

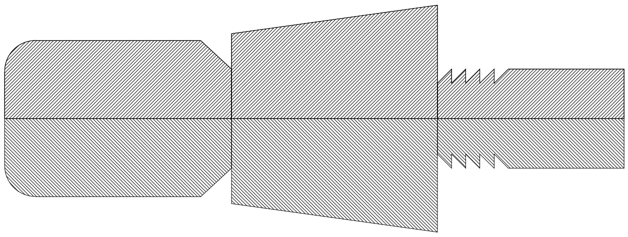

Abbildung 4.12: Leichte und schwere Featureverifikation

## 4.4 Applikationsschnittstellen

Die in dieser Diplomarbeit entwickelte Applikation ist ein Entwurfssytem für rotationssymmetrische Drehteile. Zur Modellierung von Werkstücken stehen die in Abschnitt 4.3 vorgestellten primitiven Komponenten und Features zur Verfügung. In diesem Abschnitt werden die Werkzeuge, d.h. Schnittstellen, vorgestellt, die den Konstrukteur bei seiner Aufgabe unterstützen sollen.

### 4.4.1 Modellierung eines Werkstücks

Die CAD-Applikation soll den Konstrukteur bei seiner Aufgabe unterstützen. Dafür stellt die Applikation verschiedene Werkzeuge zur Verfügung, deren Benutzung Anwendern von Graphikprogrammen bzw. CAD-Systemen und vor allem AutoCAD-Anwendern nicht fremd vorkommen wird. Wie schon in Abschnitt 4.1 erwähnt orientiert sich die Handhabung der Applikation strikt an dem darunterliegenden AutoCAD-Basissytsem.

#### Verwendung der Maus

Die „Maus“ wird zur Steuerung des Cursors, zur Auswahl von Menüpunkten und zahlreichen anderen Funktionen eingesetzt und dient so der Interaktion zwischen Konstrukteur und Applikation. Jegliche Funktionalität, die das System zur Verfügung stellt, läßt sich über die Maus aktivieren. Dabei können zwar viele Kommandos auch über die Kommandozeile eingegeben werden, doch die Maus ist das Haupteingabemedium. Die Tasten der Maus sind folgendermaßen konfiguriert worden:

- linke Maustaste: Selektion von Objekten durch einmaliges Anklicken oder Aufziehen eines Selektionsfensters durch einen Doppelklick
- mittlere Maustaste: wiederholtes Ausführen des vorherigen Kommandos
- rechte Maustaste: Aufruf eines Menüs zur Steuerung des *Object Snap* (siehe unten)

#### Modellierungsfenster

Nach dem Start der Applikation erscheint auf dem Bildschirm das Basisfenster der CAD-Applikation (siehe Abbildung 4.13). Grundlage stellt das AutoCAD-Basisfenster dar, das in Abschnitt 3.3.1 vorgestellt wurde. Dieses Fenster wurde dabei den Bedürfnissen dieser Applikation zur Erzeugung von Drehteilen angepaßt.

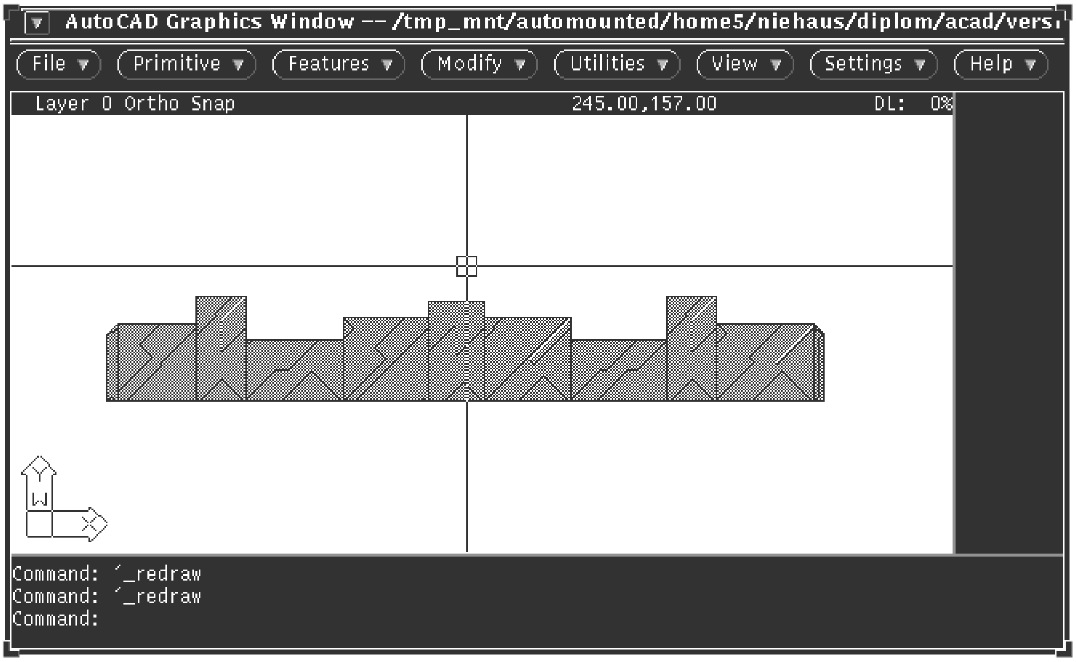


Abbildung 4.13: Applikationsfenster

### Menüs

Bei dieser Applikation kommen zwei Arten von Menüs zum Einsatz (siehe Abschnitt 3.3.1). Das sogenannte *Cursormenü*, das durch Benutzung der rechten Maustaste erscheint, ermöglicht die Steuerung des *Object Snap*, der die Art und Weise bestimmt, wie Objekte oder Punkte in einer Zeichnung selektiert werden. AutoCAD kennt dafür eine Reihe von Einstellungen, die festlegen, welcher konkrete Modus für die Koordinatenselektion mit Hilfe der Maus herangezogen wird. Z.B. läßt sich automatisch der Schnittpunkt zweier Linien mittels geeignetem Object Snap durch einfaches Klicken in der Nähe des Schnittpunktes ermitteln. Die zweite Menüart wird als Schnittstelle zur Eingabe von Kommandos verwendet. Dazu steht an der oberen Seite des Fensters eine Menüleiste mit sogenannten *Pull-Down-Menüs* zur Verfügung. Durch Positionierung des Cursors auf eines dieser Menüs und Halten der rechten Maustaste erscheint eine vertikale Menüleiste. Um einen Menüpunkt auszuwählen wird der Cursor zum gewünschten Punkt innerhalb des Menüs gezogen, wobei der selektierte Menüpunkt invers erscheint. Loslassen der Maustaste aktiviert das Kommando, das hinter dem Menüpunkt steht. Pull-Down-Menüs können dabei auch hierarchisch aufgebaut sein. Dem Benutzer von Window-Systemen sollte diese Art der Menübenutzung nicht schwerfallen.

### Formulare

*Formulare* sind Dialogfenster, die die Aufgabe haben, die Eigenschaften bzw. Parameter von den im Werkstück verwendeten primitiven Komponenten und Features anzuzeigen. Dazu müssen die entsprechenden Kommandos aktiviert werden, die dann die Selektion eines Objekts erwarten. Die beschreibenden Parameter des selektierten Objekts werden in einem Dialogfenster angezeigt. Formulare werden dahingehend unterschieden, ob sie nur beschreibenden Charakter haben, oder ob mit ihnen auch Objektänderungen zugelassen sind. Abbildung 4.14

zeigt beispielhaft Formulare für einen Zylinder und eine Abrundung. Diese Formulare eignen sich besonders zur Feinmodellierung des Werkstückes, nachdem das grobe Ausmaß des Drehteils mit der Maus spezifiziert wurde (siehe Abschnitt 4.1).

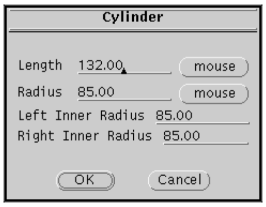



Abbildung 4.14: Beispielhafte Formulare für primitive Komponenten und Features

Weitere Formulare stehen dem Benutzer über den Menüpunkt *Settings* zur Verfügung (siehe Anhang A.5). Mit diesen Menüs werden Parameter eingestellt, die die Benutzung und das Verhalten der CAD-Applikation spezifizieren.

### 4.4.2 Modellierung von Formeinstichen

Die Geometrien der in Abschnitt 4.3 vorgestellten primitiven Komponenten und Features sind bis auf die Festlegung der Ausmaße vordefiniert. Die Ausnahme bildet hier das Feature *Formeinstich*. Wie schon bemerkt, sind den Variationen der Vertiefungsformen der Einstiche keine Grenzen gesetzt. Um dieses Potential auch für die Erstellung von Drehteilen zu erhalten, wird ein Modellierungstool für Formeinstiche zur Verfügung gestellt. Der Konstrukteur kann eigene Formeinstiche erstellen, verwalten, in Bibliotheken zusammenfassen und jederzeit in seinen Werkstücken wiederverwenden.

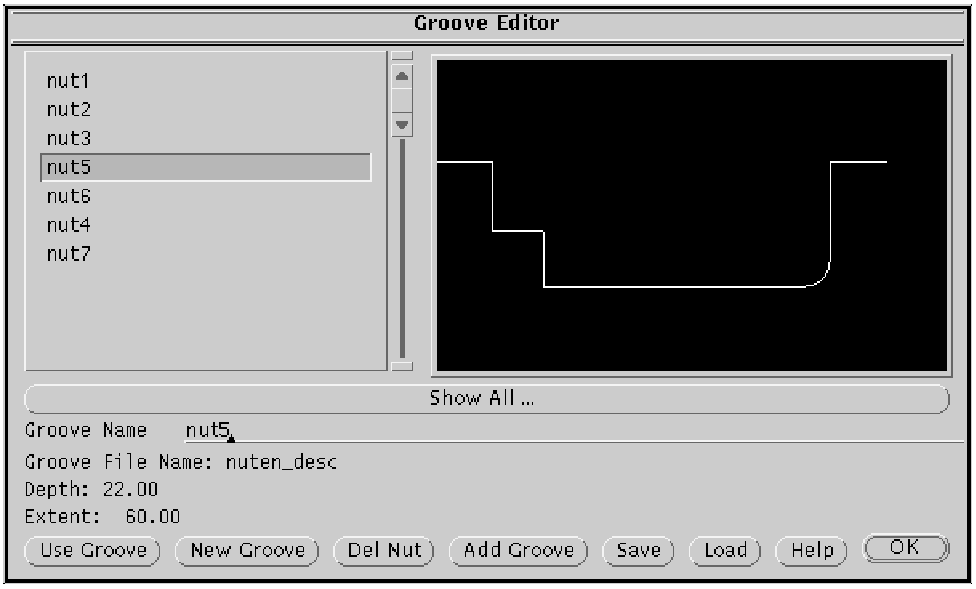


Abbildung 4.15: Nuteneditor

Die graphische Oberfläche des sogenannten *Nuteneditors* ist in Abbildung 4.15 dargestellt. Zur Erstellung eines Formeinstiches wird eine speziell an diese Aufgabe angepaßte andere

Applikation von AutoCAD verwendet, sodaß grundsätzlich jede in AutoCAD mögliche Zeichenfunktion zur Erstellung von Formeinstichen bereit steht. So ist jede nur erdenkliche Form erstellbar, mit der Bedingung, daß zwischen der linken und rechten Begrenzung des Formeinstiches ein geschlossenener Linienzug entsteht. So ergibt sich eine leichte Erweiterung der Menge der Features, indem an den primitiven Komponenten sogenannte *Innenbearbeitungen* oder *Freistiche* vorgenommen werden könnten.

### 4.4.3 Schnittstelle zu CAPlan

Aufgabe des Konstrukteurs ist die Erstellung eines Modells eines rotationssymmetrischen Drehteils. Die vorliegende CAD-Applikation soll ihn dabei durch verschiedene Werkzeuge unterstützen. Ergebnis dieses Modellierungsvorgangs ist eine Repräsentation des erstellten Werkstücks in Datenstrukturen, die nachfolgende Arbeitsprozesse verstehen und weiterverarbeiten können. Prinzipiell spielt dabei der Typ des nachgelagerten Arbeitsvorganges keine Rolle.

Im Rahmen dieser Diplomarbeit erfolgt eine Anbindung der vorliegenden CAD-Applikation an das Planungssystem CAPlan. Dazu wurde das System um eine Schnittstelle erweitert, die es CAPlan ermöglicht, Daten eines modellierten Werkstücks für seinen Planungsprozeß anzufordern. Eine Beschreibung dieser Schnittstelle erfolgt in Kapitel 5.

# Kapitel 5

# AutoCAD in einer Client-Server-Umgebung

Die vorliegende Applikation hat die Aufgabe, eine problemspezifische Repäsentation rotationssymmetrischer Drehteile zur Verfügung zu stellen. Diese Repräsentation dient nachfolgenden Arbeitsprozessen als Eingabedaten. In dieser Diplomarbeit wird das System CAPlan [Weberskirch, 1994; Weberskirch, 1995] als eines dieser nachgelagerten Arbeitsprozesse betrachtet.


Abbildung 5.1: AutoCAD und CAPlan

Abbildung 5.1 zeigt die grundlegende Struktur der Zusammenarbeit zwischen AutoCAD und CAPlan. Beide Systeme können auf verschiedenen Rechnern aktiviert sein, die natürlich über ein Netzwerk miteinander verbunden sein müssen. Mit Hilfe der sogenannten *rechnerübergreifenden Interprozeßkommunikation* ist es möglich, beide Systeme so zu koppeln, daß zwischen ihnen ein Datenaustausch durchgeführt werden kann. In Abschnitt 5.1 wird die Benutzerschnittstelle beschrieben, mit der dieser Datenaustausch durchgeführt werden kann. Dieses Kapitel soll zusätzlich einen Einblick in die *Client-Server-Umgebung* (siehe Abschnitt 5.2) geben, in der die AutoCAD-Applikation installiert wurde, und die dafür verwendeten Kommunikationsmechanismen beschreiben (Abschnitt 5.3).

## 5.1 AutoCAD → CAPlan Benutzerschnittstelle

Ein erzeugtes Werkstück wird in einer internen CAD-Struktur repräsentiert. Diese sollen als Eingabedaten für weitere Aufgaben dienen. Die nachfolgenden Arbeitsprozesse verlangen aber meistens eigene problemspezifische Datenstrukturen. Die interne Struktur eines Werkstücks ist bei der Implementation sehr allgemein gehalten, und so gibt es wenig Probleme, diese in die gewünschten Strukturen zu transformieren. Ziel dieser Diplomarbeit ist eine Schnittstelle zu erstellen, die die Bereitstellung von problemspezifischen Datenstrukturen für das System CAPlan erlaubt. Durch diese Schnittstelle bleiben die Datenstrukturen bzw. die Transformation der Datenstrukturen dem CAPlan-Benutzer verborgen.

Die AutoCAD-Applikation und das hier betrachtete System CAPlan wurden über eine Client-Server-Umgebung gekoppelt. Diese Kopplung basiert auf den Kommunikationsmechanismen der *Sockets*, die das Betriebssystem Unix bereitstellt (siehe Abschnitt 5.3). Somit müssen AutoCAD und CAPlan nicht auf demselben Rechner aktiviert werden, d.h. Konstruktion eines Werkstücks und Fertigungsplanung auf Grundlage dieses Werkstückes können zeitlich und räumlich getrennt vorgenommen werden. Dies entspricht auch dem grundlegendem Konzept des CIM, bei dem die Arbeitsvorgänge CAD, CAPP und CAM zwar getrennt ablaufen, aber über Datenaustausch eng miteinander gekoppelt sind (siehe Kapitel 1). Die hier gewählte Variante des Datenaustausches zwischen CAD-System und CAPP in einer Client-Server-Umgebung hat gegenüber anderen Konzepten, z.B. Datenaustausch über Kopieren von Dateien, den Vorteil einer gesteigerten Transparenz und Flexibilität. Zudem wird durch dieses Konzept dem Grundgedanken von AutoCAD, der *Open Architecture* (siehe Abschnitt 3.1), voll Rechnung getragen.

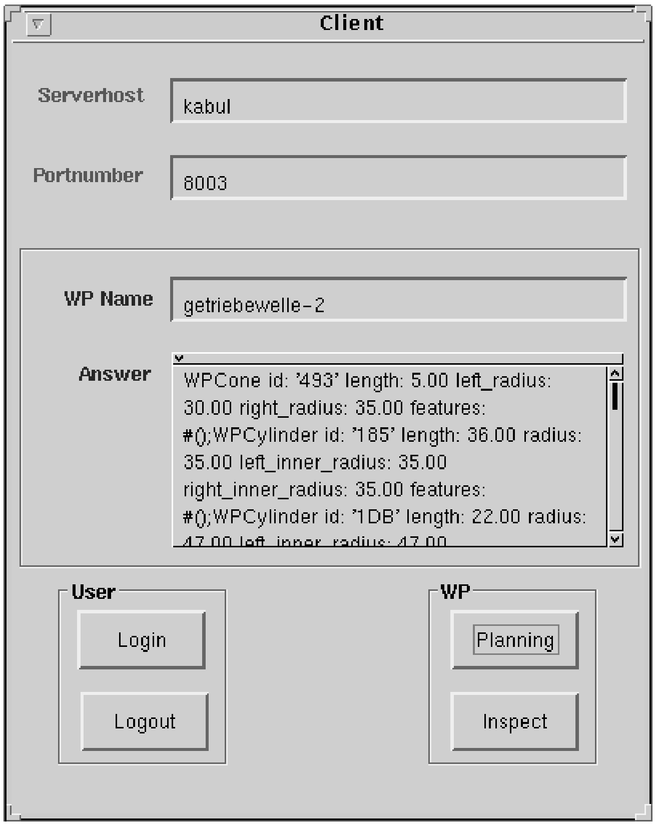


Abbildung 5.2: Client Interface

In dieser Client-Server-Umgebung stellt das Planungssystem CAPlan den Client und AutoCAD den Server dar. Das bedeutet, daß der Server vom Benutzer gestartet werden muß, um in einer aktiven Warteschleife auf Anforderungen von CAPlan zu warten. Der Server wird mit dem Menüpunkt **Export** unter dem Pull-Down-Menü **File** aktiviert. Auf der CAPlan-Seite steht ein Interface zur Verfügung, mit dem die Verbindung zu AutoCAD hergestellt und Daten bezüglich eines ausgewählten Werkstückes angefordert und weiterverarbeitet werden können (siehe Abbildung 5.2).

Folgende Punkte skizzieren den Ablauf einer Datenanforderung eines Werkstückes:

- Hostname und Portnummer des Serverrechners werden in die vorgesehenen Felder eingetragen.
- Mit der Aktivierung des Buttons **Login** loggt sich der Benutzer beim AutoCAD-Server ein. Aus Sicherheitsgründen wurde dem Server die Möglichkeit gegeben, dies auch ablehnen zu können.
- Mit der Eingabe eines Werkstücknamens in das Feld **WP Name** fordert der Benutzer die Daten des spezifizierten Werkstücks an. Nach erfolgreichem Datentransport lassen sich die Werkstückdaten durch Aktivierung des **Inspect**-Buttons anzeigen. Dieser Button wurde nur zur Demonstration installiert. Durch Aktivierung des wichtigeren **Planning**-Buttons wird der Planungsvorgangs auf den angeforderten Werkstückdaten gestartet.
- Mit dem Button **Logout** loggt sich der Benutzer beim Server wieder aus. Dieses Kommando ist besonders wichtig, da der AutoCAD-Server nur dadurch aus seiner aktiven Wartestellung befreit werden kann.

Das Feld **Answer** dient der Anzeige von Anfrageergebnissen, zeigt beispielsweise aber auch, ob das Login erfolgreich war. Die Serverportnummer gibt der Server automatisch nach seiner Aktivierung in der Kommandozeile aus.

## 5.2 Client-Server-Modell

Das Standardmodell für solche Interprozeßkommunikation, insbesondere für Netzwerkanwendungen, ist das *Client-Server*-Modell [Stevens, 1992]. Ein *Server*-Prozeß ist ein Prozeß, der darauf wartet, von einem *Client*-Prozeß angestoßen zu werden, um für den Client-Prozeß einen Dienst auszuführen. Ein typischer, aber nicht obligatorischer Ablauf besteht aus folgenden Punkten:

- Auf einem beliebigen Rechnersystem wird ein Server-Prozeß gestartet. Dieser initialisiert sich und geht anschließend in einen Wartezustand, um auf einen Client-Prozeß zu warten, für den er eine Aufgabe durchführen soll.
- Der Client-Prozeß wird entweder auf demselben System oder auf einem anderen gestartet, das über ein Netz mit dem System des Server-Prozesses verbunden ist. Der Client-Prozeß sendet über das Netz Anforderungen an den Server-Prozeß. Dieser interpretiert die Anforderung und führt entsprechende Aufgaben aus.
- Hat der Server-Prozeß seine Aufgabe für den Client-Prozeß ausgeführt, so geht er zurück in seinen Wartezustand, um weitere Client-Anforderungen zu empfangen.

Server-Prozesse lassen sich in zwei Kategorien einteilen.

**Iterative Server:** Der Server-Prozeß beantwortet jede Client-Anforderung selbst. Dies ist nur sinnvoll, wenn die Aufgabe vom Server-Prozeß innerhalb einer definierten Zeitspanne schnell erledigt werden kann.

**Parallele Server:** Ist die Zeit zur Erledigung der Aufgabe nicht vorhersehbar, weil die benötigte Zeit z.B. vom Aufruf selbst abhängt, so verfährt der Server-Prozeß auf eine andere Art. Der Server-Prozeß erzeugt einen *parallelen* Prozeß und beauftragt ihn, an seiner Stelle den Auftrag auszuführen. Der eigentliche Server-Prozeß ist somit wieder frei zur Annahme von weiteren Anforderungen von Client-Prozessen.

Man beachte, daß die Rollen von Client und Server nicht gleich sind, es tritt vielmehr ein *asymmetrischer* Zustand ein. Damit ist gemeint, daß die beiden Prozesse nicht gleich (symmetrisch) programmiert sind. Bei der vorliegenden Arbeit übernimmt die AutoCAD-Applikation die Rolle des Servers und CAPlan bzw. der nachfolgende Arbeitsvorgang die Rolle des anfragenden Clienten. Die einzige Aufgabe, die der Server-Prozeß hat, ist dem Clienten die Datenrepräsentation eines von ihm spezifizierten Werkstücks zu übertragen. Da dies in absehbarer, kurzer Zeit abgewickelt werden kann, wurde der Server-Prozeß als iterativer Prozeß implementiert.

## 5.3 Unix-Sockets zur Kommunikation

Dieser Abschnitt beschreibt das Grundkonzept der UNIX-Sockets und die verschiedenen Möglichkeiten, die sich dadurch für einen Nachrichtenaustausch ergeben.

Diese UNIX-Kommunikationsmechanismen stehen dem Programmierer durch die Implementierungssprache C [Kernighan und Ritchie, 1990] zur Verfügung. Von Smalltalk [Goldberg und Robson, 1983] aus, das Implementierungsprache für CAPlan ist, kann über eine höherliegendende Schnittstelle auf die Socket-Mechanismen zugegriffen werden.

### 5.3.1 Allgemeines zu Sockets

Eine weit verbreitete und sehr flexible Methode, eine rechnerübergreifende Interprozeßkommunikation in Form eines Client-Server Modells zu erreichen, ist die Benutzung des *Socket*-Konzeptes. Dieses Konzept realisiert eine Schnittstelle zwischen Anwendungsprogrammen und bestimmten *Kommunikationsprotokollen*. Kommunikationsprotokolle legen die erforderlichen Schritte bei der Kommunikation fest. Eine der am häufigsten verwendeten Kommunikationsschnittstellen sind die *Berkeley Sockets* [Stevens, 1992], die auch bei dieser Diplomarbeit benutzt wurden.

Bei Sockets, die, wie hier, über das Netzwerk, das Internet, angesprochen werden, spricht man von der *Internet-Domain*. Bei Sockets wird zwischen verbindungsorientierten und verbindungslosen Datenübertragungen unterschieden. Im *Internet* stehen die Transport- bzw. Kommunikationsprotokolle *TCP* (*T*ransmission *C*ontrol *P*rotocol) und *UDP* (*U*ser *D*atagram *P*rotocol) zur Verfügung. Diese unterscheiden sich im Verbindungsaufbau zwischen den kommunizierenden Systemen.

Ein *verbindungsorientierter* Nachrichtendienst setzt voraus, daß schon vor Beginn der eigentlichen Kommunikation eine logische Verbindung zwischen den beiden Teilnehmern existiert. Ein verbindungsorientierter Dienst wird oft verwendet, wenn zwischen den Systemen mehr als eine Nachricht ausgetauscht werden soll. Folgende Schritte für diesen Dienst sind erforderlich:

- Verbindungsaufbau,
- Datentransfer (auch über längere Zeit),

- Verbindungsabbau.

Der Gegensatz zum verbindungsorientierten Dienst ist der *verbindungslose* Dienst, auch mit *Datagramm*-Dienst bezeichnet. Bei diesem Dienst werden als Datagramme bezeichnete Nachrichten zwischen den Systemen übertragen. Da jede Nachricht unabhängig von einer anderen übertragen wird, muß jede Nachricht alle Informationen für den eigenen Transport enthalten, wie das Ziel der Nachrichten, die eigene Adresse und die eigentliche Nachricht. Der TCP-Protokollansatz unterstützt die verbindungsorientierte Kommunikation, während UDP eine verbindunglose Übertragung ermöglicht. Dabei garantiert der TCP-Nachrichtendienst die 100%-tige Datenübertragung zwischen Server und Client, d.h. entweder kommen die Daten komplett an oder ein Fehler wird gemeldet. Der Server und Client braucht sich also nicht darum zu kümmern, ob all Nachrichten übertragen worden sind oder nicht. Der Grund dafür ist, daß TCP verbindungsorientierte Client-Server-Modelle unterstützt, d.h. die Kommunikationsverbindung besteht schon vor der Datenübertragung.

Bei dieser Diplomarbeit wurde für den verbindungsorientierten Nachrichtenaustausch des TCP-Protokolls entschieden, da der Aufbau der Kommunikationsverbindung zwischen den Systemen konzeptuell klar und verständlich ist und zudem nach dem Verbindungsaufbau eine Reihe von Nachrichten ausgetauscht werden.

### 5.3.2 Kommunikationsaufbau

Bei dem Konzept der Sockets ist versucht worden, in Analogie zum Dateisystem die von dort bekannten Systemaufrufe zu übertragen. Dabei sind bei einer Kommunikation über ein Netzwerk die Sachverhalte komplexer als beim Dateisystem. Man denke an das unsymmetrische und kommunikative Verhalten von Client und Server, oder an die Unterscheidung zwischen verbindungsorientierter und verbindungsloser Datenübertragung. Die Verbindung zwischen den Kommunikationspartnern ist durch mehr Parameter als nur den Dateinamen zu beschreiben.

Eine Prozeßverbindung im Internet ist ein 5er-Tupel der folgenden Form:

$$(\text{Protokoll, lokaler Host, lokaler Port, ferner Host, ferner Port}).$$

Ein Socket stellt dabei eine Halbverbindung dar und spezifiziert folgende Parameter des obigen 5er-Tupels.:

$$(\text{Protokoll, Host, Port}).$$

*Host* steht für die jeweilige Internet-Adresse des Host, und *Protokoll* legt das Transportprotokoll fest. Anschaulich ist ein Socket das Ende einer rechnerübergreifenden Pipe. Mit Sockets wird eine byteorientierte Interprozeßkommunikation realisiert. Wie bei Pipes werden unstrukturierte Bytefolgen übertragen. Im Gegensatz zu Pipes sind Sockets jedoch bidirektional. Wenn ein Prozeß eine Pipe einrichtet, erhält er zwei File-Deskriptoren, einen für die Lese- und einen für die Schreibeseite der Pipe. Richtet er ein Socket ein, dann erhält er lediglich einen Socket-Deskriptor über dem lesender und schreibender Zugriff möglich ist. Bei einer Prozeßverbindung muß bei beiden Sockets stets dasselbe Protokoll eingestellt sein.

Um die fünf Komponenten einer Prozeßverbindung zu spezifizieren, stehen eine Reihe von Socket-Systembefehlen zur Verfügung. Abbildung 5.3 zeigt eine typische TCP-Kombination

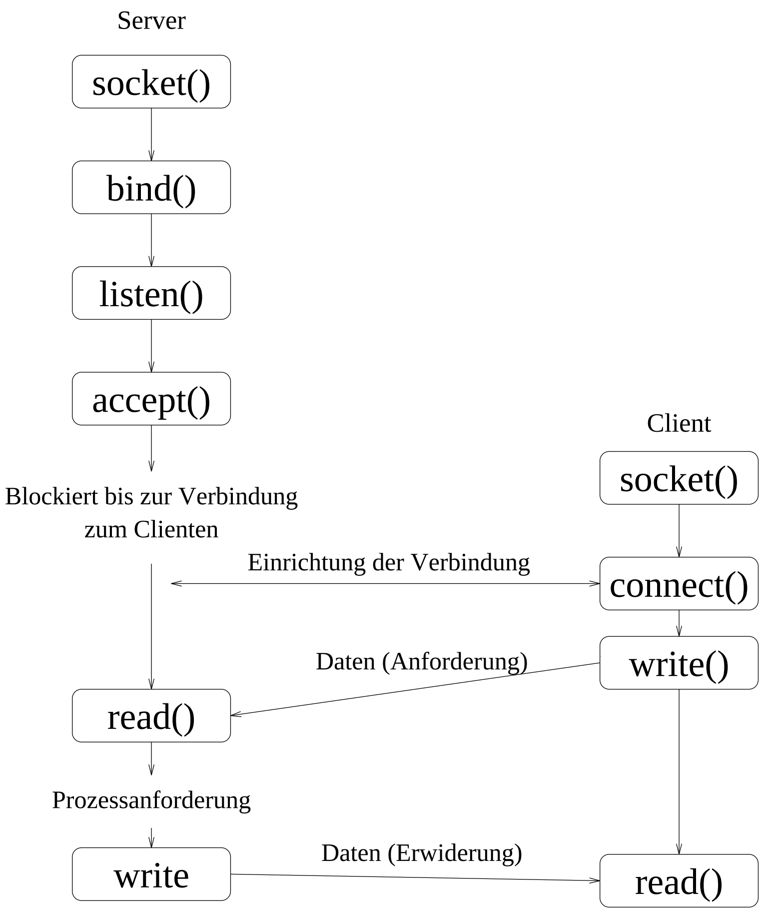


Abbildung 5.3: Typisches Kommunikationsprotokoll

von Aufrufen, die den Kommunikationaufbau zwischen Client und Server realisieren. Die Reihenfolge dieser Aufrufe ist dabei vom gewählten Transportprotokoll abhängig.

Der `socket()`-Aufruf selbst legt nur das Transportprotokoll fest. Dies erfolgt sowohl beim Server als auch beim Client. Dieser Aufruf liefert einen *Socketdeskriptor* zurück, der bei jedem weiteren Socketaufruf benötigt wird. Mit dem `bind()`-Aufruf wird der lokale Host und der lokale Port festgelegt, d.h. der zu erzeugende Server wird an diesen Port und an den lokalen Host gebunden. Mit Hilfe des `listen()`-Befehls kann festgelegt werden, auf wieviele parallele Client-Anforderungen der Server gleichzeitig *hören* soll. Durch das `accept()`-Kommando wird der Server in eine Wartestellung gebracht, in der er auf eine aktuelle Verbindungsaufforderung eines Clienten wartet. Das *Rendezvous* von Client und Server, d.h. die eigentliche Verbindung kommt erst zustande, wenn der Client mit einem `connect()`-Aufruf seinen Verbindungswunsch äußert. Zu beachten ist hier die *Asymmetrie* der Kommandos accept() und connect(), d.h. eine Verbindung kommt nur zustande, wenn der accept()-Aufruf zeitlich vor dem connect()-Aufruf erfolgt. Nach erfolgreichem Verbindungsaufbau kann mit `read()`- und `write()`-Aufrufen die Datenübertragung erfolgen. Dabei muß natürlich jedem write() auf einer Seite ein read() auf der anderen Seite entsprechen, und umgekehrt.

Der in Abbildung 5.3 vorkommende Begriff *Prozeßanforderung* deutet an, daß das in Abschnitt 5.2 vorgestellte Konzept des *parallelen Servers* auch hier implementiert werden kann, indem ein neuer Prozeß initiiert wird, der die vom Server auszuführende Aufgabe erledigen soll.

Die hier genannten Systemaufrufe beziehen sich auf eine C-Implementation des Socket-Konzeptes. In Smalltalk wird die gleiche Funktionalität über eine höher liegende Schnittstelle dem Benutzer angeboten, so daß das in obiger Abbildung dargestellte konzeptuelle Szenario nicht verändert werden muß.

## 5.4 Programmierschnittstellen

In Abschnitt 5.1 wurde die Schnittstelle zu CAPlan vorgestellt. Dies beinhaltete eine Erläuterung des grundlegenden Konzeptes, der Darstellung und der Benutzung des Oberflächeninterfaces, das entwickelt wurde, um Drehteildaten, die mit der AutoCAD-Applikation generiert wurden, für CAPlan verfügbar zu machen. In diesem Abschnitt soll ein etwas detaillierter Einblick in den strukturellen Aufbau dieser Schnittstelle erfolgen. Abbildung 5.4 zeigt diesen strukturellen Aufbau.

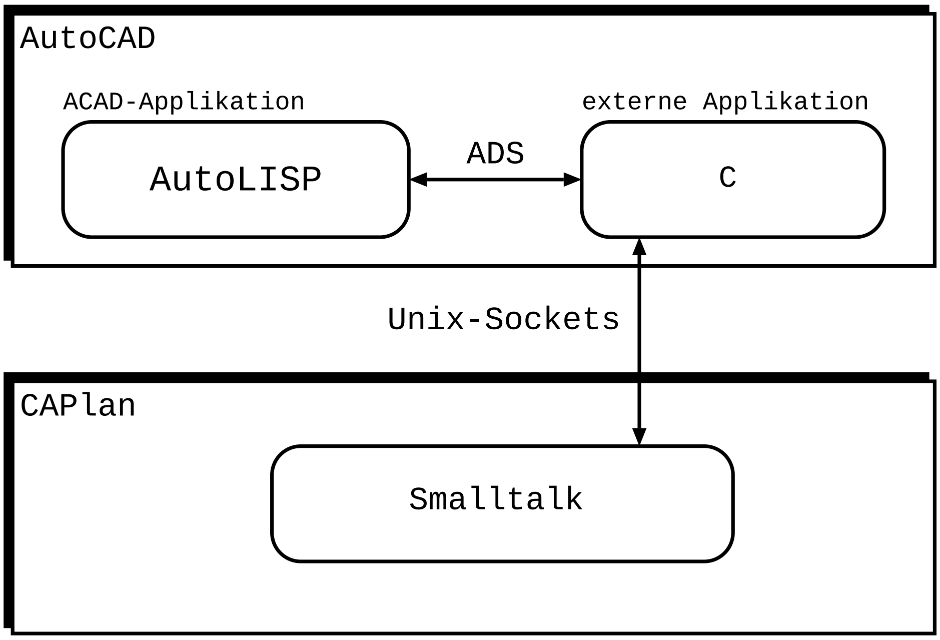


Abbildung 5.4: Schnittstellen zwischen AutoCAD und CAPlan

Die Applikation zur Erstellung von Drehteilen ist in AutoLISP implementiert. Von AutoLISP ist es nicht direkt möglich, die UNIX-Betriebssystemmechanismen zur Netzwerkkommunikation zu nutzen, da diese Mechanismen in C implementiert sind. Aber mit Hilfe der von AutoCAD bereitgestellten ADS-Schnittstelle können in C programmierte Funktionen bzw. Applikationen, die diese Funktionen vereinen, so an AutoLISP gebunden werden, daß diese *externen Applikationen* als AutoLISP-Funktionen betrachtet und auch als solche aufgerufen werden. Diese Möglichkeiten, die das ADS-Paket bietet, wurden in Abschnitt 3.3.2 beschrieben. Erst von der externen Applikation kann man das UNIX-Socketkonzept benutzen, um mit CAPlan Daten auszutauschen.

Die Beziehung zwischen AutoLISP und einer externen C-Applikation ähnelt der Beziehung zwischen AutoCAD und CAPlan, denn beiden liegt das Client-Server-Modell zugrunde. Die externe Applikation stellt gewissermaßen den Serverprozeß dar, der auf eine Aufforderung von AutoLISP wartet, eine seiner externen Funktionen auszuführen. Nach Ausführung der spezifizierten Funktion kann ein Rückgabewert an AutoLISP übergeben werden. AutoLISP repräsentiert in dieser Beziehung den Clienten (siehe Abschnitt 3.3.2). Der Unterschied zur Client-Server-Beziehung zwischen der ACAD-Applikation und CAPlan ist, daß der Client AutoLISP und die externe Applikation als Server auf demselben Rechner ablaufen. Es handelt sich also hier um eine *rechnerlokale Interprozeßkommunikation*.

Auch für diese lokale Interprozeßkommunikation muß ein bestimmtes *Protokoll* eingehalten werden, wie dies auch für die Kommunikation über Sockets zwingend erforderlich ist. Dieses Protokoll soll nun skizziert werden.

AutoLISP lädt die ADS-Applikation mit der Funktion `(xload "Applikationsname")`. Daraufhin wird der hinter diesem Namen stehende Server aktiviert. Dieser hat folgenden Aufbau.

```
#include  "/agr.local/acad/ads/adslib.h"

char *buffer;
struct resbuf *value;
struct func_entry { char *func_name; int (*func) _((void)); };

/* externe Funktionen werden deklariert */
int start_server     _((void));
int wait_for_client  _((void));
int destroy_server   _((void));
int send_to_client   _((struct resbuf *rb));

/* externe Funktionen erhalten ihren LISP-Funktionsnamen*/
static struct func_entry func_table[]
          = { {"start_server", start_server},
              {"wait_for_client", wait_for_client},
              {"destroy_server", destroy_server},
              {"send_to_client", send_to_client},
                   };

/* lokale Funktionen */
void     main       _((int, char **));
int      dofun      _((void));
int      funcload   _((void));

/*-----------------------------------------------------------------------*/
/* MAIN - das Hauptprogramm */

void main(argc,argv)
{
  short scode = RSRSLT;   /* Ergebniscode (default) */

  ads_init(argc, argv);   /* Initiiert Kommunikation zu AutoLISP */

  for ( ;; ) {            /* Anfrage/Ergebnis-Schleife */

      if ((stat = ads_link(scode)) < 0)
```

```
            sprintf(errmsg, "Error form ads_link()",stat);
        }

        switch (stat) {

        case RQXLOAD: /* externe Funktionen werden geladen und definiert */
            scode = funcload() == RTNORM ? RSRSLT : RSERR;
            break;

        case RQSUBR:  /* aufgerufene externe Funktion wird aktiviert */
            scode = dofun() == RTNORM ? RSRSLT : RSERR;
            break;

        default:
            break;
        }
    }
}
```

Nach dem `(xload)` wird die Kommunikationsverbindung zu AutoLISP mit dem Aufruf `ads_init()` aufgebaut. Durch `ads_link()` signalisiert die ADS-Applikation, daß sie bereit ist, eine Funktionsanforderung von AutoLISP entgegenzunehmen. Anhand der `stat`-Variable erkennt der Server, ob er die externen Funktionen noch laden und definieren muß (mit `funcload()`), oder ob eine von AutoLISP aufgerufene Funktion ausgeführt werden soll (mit `dofun()`).

In der Funktion `funcload()` werden die externen Funktionen geladen und definiert. Mit Hilfe der Tabelle `func_table` werden die LISP-Namen der externen Funktionen an die Namen der entsprechenden externen C-Funktionen gebunden. Dies ermöglicht eine flexible Zuordnung von LISP-Funktionsnamen und externen Funktionen. In diesem Beispiel sind diese aber identisch.

Erfolgt von AutoLISP aus ein Aufruf einer externen Funktion, so wird die lokale Funktion `dofun()` aktiviert, die dann anhand der Tabelle `func_table` die entsprechende C-Funktion ausführt. Hier ist auch eine Wertrückgabe an AutoLISP möglich.

Mit dem Kommando `(xunload "Applikationsname")` auf der AutoLISP-Seite und dem Aufruf `ads_xunload "Applikationsname"` auf der C-Seite wird die Verbindung zur externen Applikation beendet. Dies befreit den Server aus seiner Wartesituation.

Die Funktionalität der externen Funktionen entspricht gewöhnlichen C-Funktionen. Der folgende Programmcode implementiert obige externe Funktion `destroy_server()`.

```
static int destroy_server ()
{
  ads_xunload ("start_server");
  return RTNORM; /* Default-Rueckgabewert */
}
```

Die ADS-Schnittstelle ist sehr effektiv und sinnvoll, da nur so die Möglichkeit besteht, Betriebssystemmechanismen von AutoLISP aus zu benutzen.

# Kapitel 6

# Zusammenfassung und Ausblick

Im Rahmen dieser Diplomarbeit wurde ein Konstruktionswerkzeug realisiert, mit dem rotationssymmetrische Drehteile *modelliert* werden können. Aus dieser Darstellungsform werden Problemspezifikationen für die Arbeitsplanerstellung für Werkstücke dieser Objektdomäne generiert. Dem Planungssystem CAPlan [Weberskirch, 1994; Weberskirch, 1995] dienen diese Spezifikationen als Eingabedaten für das nachgelagerte Problem der Fertigungsplanung. Ausdrückliche Ziele bei der Realisierung waren ein geringer Zeitaufwand beim Modellieren der Drehteile, ein hoher Modellierungs- und Bedienungskomfort des Werkzeugs und eine Anbindung an das Planungssystem CAPlan. Das zu entwickelnde Konstruktionswerkzeug sollte unter Verwendung eines kommerziellen, in der Industrie verbreiteten CAD-Systems entwickelt werden, um einen möglichst hohen Akzeptanzgrad zu erzielen.

In einer früheren Diplomarbeit wurde schon ein Werkzeug mit ähnlichen Zielen und Aufgaben entwickelt. Dort wurde das System *GraMoD* [Kiefer, 1992] in der Programmiersprache Smalltalk implementiert. Die Entscheidung für Smalltalk und damit auch für dessen geringe Graphikunterstützung hatte die Konsequenz, daß jegliche graphische Darstellung und Benutzerinteraktion selbst programmiert und entwickelt werden mußte, mit dem Ergebnis, daß unter kommerziellen Gesichtspunkten der Einsatz des Systems eher fragwürdig sein mußte.

In der hier vorliegenden Diplomarbeit wurde ein etwas anderer Ansatz gewählt: Entwicklung einer CAD-Applikation zur Modellierung von rotationssymmetrischen Drehteilen aufbauend auf der schon vorhandenen Funktionalität eines CAD-Basissystems. Als CAD-Basissystem wurde das AutoCAD-System Release 12 der Firma AutoDesk ausgewählt. Zur Entwicklung von problemspezifischen CAD-Applikationen auf der Basis eines kommerziellen CAD-Systems ist das AutoCAD-System gut geeignet. Es ermöglicht durch die Programmiersprachen C und AutoLISP eine effiziente Applikationsentwicklung. Von diesen Sprachen aus läßt sich auf jede Funktionalität zugreifen, die das Basissystem bereit stellt. Damit steht schon ein breit gefächertes und großes Angebot an Zeichen- bzw. Modellierungsfunktionen zur Verfügung, das bei der Entwicklung von eigenen CAD-Applikationen genutzt und erweitert werden kann. Dieses Angebot an Funktionalität schließt auch Methoden zur Benutzerinteraktion ein. Mit dem Ansatz des CAD-Basissystems als Implementierungsgrundlage wurde ein professionelles Aussehen und eine professionelle Handhabung des entwickelten Werkzeugs bei gleichzeitig geringem Zeitaufwand des Modellierungsprozesses errreicht. Dieses Ergebnis steht zudem einem vertretbaren Entwicklungsaufwand gegenüber. Beides konnte im System GraMoD nicht in der Weise realisiert werden. Aus diesen Erfahrungen hat man gelernt, daß Eigenentwicklungen einen weitaus geringeren Akzeptanzgrad haben als kommerzielle Produkte.

Leider steht mit AutoCAD Release 12 keine objektorientierte Programmiersprache zur Verfügung. Dies wäre, wie schon in [Kiefer, 1992] erwähnt, zur Modellierung von realen Objekten sehr effektiv und naheliegend. In diesem Punkt hat das System GraMod mit der objektorientierten Sprache Smalltalk den Vorteil der natürlicheren Repräsentation von realen Sachverhalten. Objektorientierung ist für AutoCAD erst für die Version 13 vorgesehen.

Im Rahmen der computerintegrierten Fertigung (CIM) ist der Modellierungsprozeß zur Erstellung von Problemspezifikationen für technische Werkstücke mit Hilfe des CAD unabhängig von der nachfolgenden Fertigungsplanung (CAPP) [Roller, 1995; Encarnacão und Schlechtendahl, 1983]. Aus diesen Überlegungen heraus sollte die Anbindung eines CAD-Werkzeugs zur Werkstückmodellierung an CAPlan so vorgenommen werden, daß der Modellierungvorgang und die Fertigungsplanung räumlich und zeitlich voneinander getrennt werden. Da CAPlan und das System GraMoD in Smalltalk implementiert sind, erfolgt dort die Kopplung, indem beide Systeme im selben Image ablaufen. Den obigen Überlegungen wird dadurch keine Beachtung geschenkt. Um dies zu verbessern, wurde bei der vorliegenden Arbeit eine *Client-Server-Umgebung* geschaffen, in der das Modellierungswerkzeug die Rolle des Servers und das Planungssytem CAPlan die Rolle des Clienten einnimmt. Somit können Modellierungsaufgabe und Planungaufgabe bezüglich eines Werkstücks auf verschiedenen Rechnern und von verschiedenen Personen ausgeführt werden, und stellen damit jeweils unabhängig voneinander laufende Prozesse dar. Die Client-Server-Beziehung wird mit Hilfe der Kommunikationsmechanismen der *Berkeley Sockets* des zugrundeliegenden Betriebssystem UNIX aufgebaut. Die Schnittstelle zu diesem Mechanismus ist durch die Programmiersprache C gegeben, die vom System AutoCAD dem Applikationentwickler zur Verfügung gestellt wird.

In bezug auf die bisher modellierbare Objektdomäne und auf die nachgelagerte Fertigungsplanung werden durch die Begriffe *Flexibilität* und *Universalität* weitere wichtige Ziele beschrieben. Um hohe Flexibilität zu gewährleisten, sind die internen Datenstrukturen, die das zu modellierende Werkstück in seiner Geometrie repräsentieren, sehr einfach gehalten. Da diese ausnahmslos LISP-Strukturen sind, wurden auf komplizierte und komplexe Strukturen und Beziehungen zwischen ihnen verzichtet. Dies kommt einer leichten Erweiterung der bisher möglichen primitiven Komponenten und Features entgegen. Durch diesen Implementierungsgrundsatz läßt sich die Werkstückspezifikation, die sich als Ergebnis des Modellierungsprozesses ergibt, sehr universell den speziellen Anforderungen der Fertigungsplaner anpassen. Somit kann das vorliegende Modellierungswerkzeug auch für andere Planungssysteme Problemspezifikationen erstellen. Zusätzlich brauchen die schon einmal modellierten Drehteile nicht erneut erstellt zu werden.

Bei den Kommunikationsmechanismen wurde bereits darauf geachtet, daß eine Anbindung an das System *CoMo-Kit* (*Conceptual Model Construction Kit*, [Maurer, 1993; Maurer und Pews, 1996]) mit geringem Aufwand möglich ist. Das System CoMo-Kit unterstützt die Planung und Durchführung komplexer, verteilter Entwurfsprozesse. Die Prozesse der computerintegrierten Fertigung (CIM) könnten also mit dem System CoMo-Kit gesteuert und koordiniert werden. Dies schließt den Bereich des CAD, für den das vorliegende Werkzeug entwickelt wurde, ein. Aus diesem Grund wurden in dieser Arbeit Kommunikationsmechanismen implementiert, die eine Anbindung an CoMo-Kit ermöglichen.

# Anhang A

# Applikationsbenutzung

Dieser Anhang beinhaltet eine Übersicht zur Benutzung des im Rahmen dieser Diplomarbeit entwickelten Entwurfsystems für rotationssymmetrische Drehteile. Abbildung A.1 zeigt die Menüleiste des Systems als einen Auschnitt der Gesamtbenutzeroberfläche (siehe Abbildung 4.13). Die Benutzung des Entwurfsystems wird anhand dieser Menüleiste skizziert, indem die Funktionalität beschrieben wird, die hinter den einzelnen Menüpunkten der Menüleiste steht.

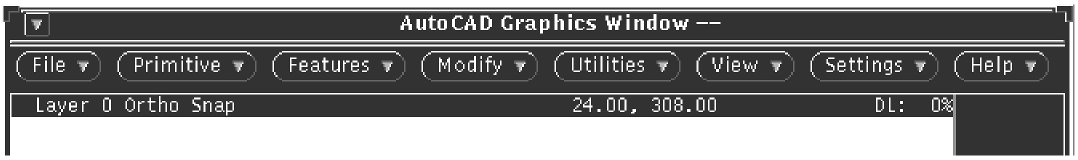


Abbildung A.1: Menüleiste

## A.1 File

Das `File`-Menü faßt alle Funktionen zusammen, die zum Laden, Speichern und Exportieren von Werkstücken benötigt werden.

**New** generiert eine neue unbenannte Zeichenfläche.

**Load** lädt ein existierendes Werkstück.

**Save** speichert das aktuelle Werkstück.

**Save As** speichert das aktuelle Werkstück unter einem zu spezifizierenden Namen.

**Export** baut eine Client-Server-Verbindung zur Übertragung von Werkstückdaten zu CAPlan auf.

**Exit to AutoCAD** beendet die Drehteilapplikation und startet das darunterliegende AutoCAD-Basissystem.

**Quit** beendet sowohl die Drehteilapplikation als auch das darunterliegende AutoCAD-System.

## A.2 Primitive

Dieser Menüpunkt dient der Ergänzung und Veränderung von primitiven Werkstückkomponenten.

**Add/Add left** erlaubt die Erweiterung des aktuellen Werkstücks um eine primitive Komponente. Es gibt drei Arten von primitiven Komponenten (siehe Abschnitt 4.3.2):

- Cylinder
- Cone
- Toroid

Ein Werkstück kann auf zwei Arten um eine primitive Komponente erweitert werden, erstens indem mit Hilfe des Menüpunkts *Add* eine Komponenten rechts an das bestehende Werkstück angehängt wird, zweitens indem links von einer existierenden Komponente mit dem Menüpunkt *Add left* eine neue Komponente plaziert wird.

Die Art und Weise wie eine primitive Komponente erzeugt wird ist bei allen drei Komponenten sehr ähnlich. Die Unterscheidung liegt nur in mehr oder weniger erforderlichen Parametern für die zu erzeugende neue Komponente.

**Show Properties** öffnet ein Formular, das alle Parameter bezüglich der selektierten Komponente anzeigt (siehe Abschnitt 4.4.1).

**Change Properties** zeigt in einem Formular alle Parameter bezüglich der selektierten primitiven Komponente. Mit Hilfe des Formulars sind Parameteränderungen der entsprechenden Komponente möglich (siehe Abschnitt 4.4.1). Diese Änderungen können sowohl mit Hilfe der Tastatur als auch mit Hilfe der Maus vorgenommen werden.

## A.3 Features

Unter diesem Menüpunkt werden alle Funktionen zusammengefaßt, die der Erzeugung, Entfernung und Anzeige von Features dienen.

**Add** erlaubt die Verfeinerung der selektierten primitiven Komponente, indem diese Komponente um ein Feature erweitert wird. Es gibt vier Arten von Features (siehe Abschnitt 4.3.3):

- RoundOff
- Slope
- Threat
- Groove

**Remove** entfernt das ausgewählte Feature von der entsprechenden Komponente. Diese Option ist sehr wichtig, da es keine Möglichkeit gibt, die Parameter von bestehenden Features zu ändern.

**Show** öffnet ein Formular, das alle Parameter bezüglich des selektierten Features anzeigt.

## A.4 Modify

Das `Modify`-Menü umfaßt Funktionen zur Veränderung des aktuellen Werkstücks.

**Remove Primitive** entfernt eine selektierte primitive Komponente mitsamt ihren Features aus dem aktuellen Werkstück.

**Move Workpiece** ermöglicht die Positionierung des aktuellen Werkstücks innerhalb der Zeichenfläche mit Hilfe der Maus. Dies ist sinnvoll, wenn z.B. der Platz an einer Seite des Werkstücks nicht mehr für eine neue primitive Komponente ausreicht.

## A.5 Utilities, View, Settings

**Show Whole Workpiece** zeigt zusätzlich zu der oberen Hälfte des aktuellen Werkstücks auch seine untere Hälfte, d.h. das komplette Werkstück wird dargestellt.

**Groove Editor** ruft den *Nuteneditor* zur Modellierung und Verwaltung von Formeinstichen auf (siehe Abschnitt 4.4.2).

**Redraw** nimmt eine graphische Aktualisierung bzw. Verbesserung des aktuellen Werkstücks vor. Dies kann nötig werden, wenn AutoCAD z.B. nach *Remove Primitive* kleine *Punkte* auf dem Bildschirm hinterläßt.

**Zoom** erlaubt das *Zoomen* der aktuellen Werkstückzeichnung. Dabei stehen verschiedene Modi zur Verfügung, wie z.B. Zoomen bezüglich eines Fensterausschnitts oder maximales bzw. minimales Zoomen des Werkstücks.

**Drawing Aids** öffnet ein Dialogfenster mit deren Hilfe das optische Verhalten des Systems festgelegt werden kann.

**Object Snap** öffnet ein Dialogfenster mit der der *Object Snap* (siehe Abschnitt 4.4.1) dauerhaft eingestellt und die Größe der *Targetbox* festgelegt werden kann.

**Selection Settings** öffnet ein Dialogfenster mit deren Hilfe verschiedene Modi zur Selektion eines Entities ausgewählt werden können. Außerdem läßt sich hiermit die Größe der *Pickbox* einstellen.

## A.6 Help

**Help** aktiviert eine On-line Hilfe, die Informationen zu den Befehlen und Kommandos des vorliegenden Konstruktionswerkzeugs bereitstellt. Diese Hilfe kann entweder mit dem Kommando *help* in der Kommandozeile oder durch Aktivierung des *Help* Buttons in der Menüleiste aufgerufen werden. Diese Aktivierung des Buttons ist zu jeder Zeit, d.h. auch bei einer Eingabeaufforderung, möglich. Dies ist sinnvoll, um auch während der Modellierung des Werkstücks Erklärungen zu Eingabeparametern zu erhalten.

# Anhang B

# Implementierungsstruktur

In diesem Anhang wird die Implementierungsstruktur, d.h. die erforderlichen Dateien und die zugrundeliegende Verzeichnisstruktur, skizziert. Um den Aufbau dieser Struktur darstellen zu können, wird exemplarisch ein Verzeichnis `acad_application` vorausgesetzt. Dort sind folgende Verzeichnisse und Dateien nötig:

**inits:** Diese Datei setzt die Unix-Systemvariablen für das System AutoCAD. Die Datei muß vor dem Starten der Applikation mit dem Kommando `source inits` aufgerufen werden.

**config/:** Sie beinhaltet die Konfigurationsdateien für die Rechner, von denen aus AutoCAD bzw. die vorliegende AutoCAD-Applikation gestartet werden kann.

**source/:** Dieses Verzeichnis beinhaltet die Quellcode-Dateien der Applikation.

**works/:** Unter diesem Verzeichnis werden die erstellten Werkstücke verwaltet.

**nuten/:** Unter diesem Verzeichnis werden die Formeinstiche und ihre Zusammenfassung in Bibliotheken verwaltet.

**dummy_template.dwg:** Dies ist eine Hilfsdatei, mit der der Pfad zu den benötigten Quellcode-Dateien und Werkstücken bestimmt wird.

Die folgenden Unix-Kommandos zeigen beispielhaft den Aufbau der Datei `inits`:

```
setenv ACAD ".../acad_application;
             .../acad_application/source;
             .../acad_application/nuten;
             .../acad_application/works;
             /agr.local/acad/acad;
             /agr.local/acad/support;
             /agr.local/acad/fonts;"

setenv ACADCFG .../acad_application/config
setenv ACADDRV /agr.local/acad/drv
```

Die folgende Liste beinhaltet die Sourcecode-Dateien, die unter dem Verzeichnis `source/` verwaltet werden.

**acad.lsp** wird jedesmal automatisch aufgerufen und der entsprechende Sourcecode evaluiert, wenn AutoCAD gestartet oder eine neue Zeichnung geladen wird. Mit Hilfe dieser Datei werden Initialisierungen für die Applikation vorgenommen. Von dieser Datei aus wird die Datei *init.lsp* geladen, die die benötigten Sourcecode-Dateien lädt.

**init.lsp** lädt die für die Applikation notwendigen Sourcecode-Dateien.

**acad_server.lsp** beinhaltet LISP-Funktionen, die für die Übertragung von Werkstückdaten zu CAPlan nötig sind. Diese Funktionen realisieren in der Client-Server-Umgebung von AutoCAD und CAPlan den Server.

**acad_server.c** beinhaltet C-Funktionen, die in der Client-Server-Umgebung von AutoCAD und CAPlan den Clienten realisieren und so für den Empfang von Werkstückdaten verantwortlich sind.

**change_obj.lsp** beinhaltet Funktionen zur Änderung von Parametern der primitiven Komponenten.

**create_obj.lsp** beinhaltet Funktionen zur Erzeugung von neuen primitiven Komponenten.

**draw_obj.lsp** beinhaltet Funktionen, die die primitiven Komponenten zeichnen.

**features.lsp** beinhaltet Funktionen zur Erzeugung, Anzeige und Entfernung von Features.

**load_save.lsp** beinhaltet Funktionen zum Laden und Speichern der Werkstücke und deren Datenrepräsentation.

**nuten.lsp** beinhaltet Funktionen zur Erzeugung und Verwaltung von Formeinstichen.

**dialog.dcl bzw. dialog.lsp** beinhaltet Funktion für das Aussehen bzw. für Verhalten der Dialogfenster.

**error.lsp** beinhaltet Funktionen zur Behandlung von Benutzerunterbrechungen, die mit der Tastenkombination `CTRL-C` möglich sind.

**exit.lsp** beinhaltet Funktionen zum Verlassen der Drehteilapplikation und zum erneuten Aktivieren.

**modify_obj.lsp** beinhaltet Funktionen zum Entfernen von primitiven Komponenten aus dem aktuellen Werkstück.

**move_obj.lsp** beinhaltet Funktionen zum Verschieben von primitiven Komponenten.

**undefine.lsp** beinhaltet Kommandos, die AutoCAD-Kommandos durch applikationspezifische Kommandos überdefinieren.

**show.lsp** beinhaltet Funktionen zur Aktivierung von Formularen, die zur Darstellung von Parametern der primitiven Komponenten dienen, und Funktionen zur Darstellung des kompletten Werkstücks.

**status.lsp** beinhaltet Initialisierungen von globalen Variablen.

**structure.lsp** beinhaltet Funktionen, die die Datenstrukturen für die primitven Komponenten und Features erzeugen.

**user_coordinate.lsp** beinhaltet Funktionen zur Erzeugung und Verwaltung des Benutzerkoordinatenkreuzes.

**user_util.dcl bzw. user_util.lsp** beinhaltet Funktionen für das Aussehen bzw. für Verhalten des Dialogfensters *Drawing Aids*, das unter dem Menüpunkt *Settings* verfügbar ist.

**verwaltung.lsp** beinhaltet Funktionen zur Verwaltung der benötigten Datenstrukturen, d.h. Zugriffs-, Lösch- und Transformationsfunktionen auf Listen und auf semantische Objekte.

**acad.hlp** beinhaltet Informationen, die über die *On-line Hilfe* der Applikation abgerufen werden können.

**proto.mnu** beinhaltet einen speziellen Code, der den Aufbau der Applikationsoberfläche spezifiziert. Wird ein Werkstück geladen oder soll eines neues generiert werden, dann wird automatisch diese Datei als Beschreibungsinformation der Applikation zugrundegelegt.

**Client bzw. ClientInterface** sind Smalltalk-Klassen, unter denen die Implementierung des *Client Interface* (siehe Abschnitt 5.1) verfügbar ist.

# Literaturverzeichnis


AUTODESK. 1992a. *ADS Programmers Reference Manual.* Autodesk Inc.

AUTODESK. 1992b. *Advanced Modeling Extension.* Autodesk Inc.

AUTODESK. 1992c. *AutoCAD Interface, Installation and Performance Guide - SUN Sparc.* Autodesk Inc.

AUTODESK. 1992d. *AutoCAD Reference Manual.* Autodesk Inc.

AUTODESK. 1992e. *AutoLISP Programmers Reference.* Autodesk Inc.

AUTODESK. 1992f. *SQL Extension Reference Manual.* Autodesk Inc.

BERNADI, A., KLAUCK, C. UND LEGLEITNER, R. 1991a. *TEC-REP: Repräsentation von Geometrie- und Technologieinformationen.* Dokument D-91-07. Deutsches Forschungszentrum für Künstliche Intelligenz GmbH.

BERNADI, A., KLAUCK, C. UND LEGLEITNER, R. 1991b. *TEC-REP: Representing Features in CAD/CAM.* Research Report RR-91-20. Deutsches Forschungszentrum für Künstliche Intelligenz GmbH.

CUNNINGHAM, J.J. UND DIXON, J.R. 1994. Proceedings of the CSG 94 Conference. *In: Set-theoretic Solid Modelling Techniques and Applications*, Band Vol. 1.

ENCARNACÃO, J. UND SCHLECHTENDAHL, E.G. 1983. *Computer Aided Design.* Springer-Verlag.

GOLDBERG, ADELE UND ROBSON, DAVID. 1983. *SMALLTALK-80: The Language and its Implementation.* Addison-Wesley.

GOSSARD, D.C. UND SAKURAI, H. 1990. Recognizing Shape Features in Solid Models. *IEEE Computer Graphics and Applications*, pp. 22–32.

KAMP, G. 1991. *Entwurf, vergleichende Bewertung und Integration eines Arbeitsplanerstellungssystems für Drehteile.* Dokument D-91-06. Deutsches Forschungszentrum für Künstliche Intelligenz GmbH.

KEANE, M., HALTON, J.P. UND MANAGO, M. (Hrsg.). 1995. *Controlling non-linear hierarchical planning by case replay.* Lecture Notes in Artificial Intelligence, Nr. 984. Springer.

KERNIGHAN, BRIAN W. UND RITCHIE, DENNIS M. 1990. *Programmieren in C.* 2. Ausg. Prentice-Hall.

KETTNAKER, V. 1995. *Konzeption und Realisierung einer Toolbox statischer Kontrollmethoden zur Steuerung eines Causal Link Planers.* Diplomarbeit, Universität Kaiserslautern.

Kiefer, A. 1992. *Ein objektorientiertes, featurebasiertes Entwurfssystem für rotationssymmetrische Drehteile*. Diplomarbeit, Universität Kaiserlautern.

Maurer, F. 1993. *Hypermediabasiertes Knowledge-Engineering für verteilte wissensbasierte Systeme*. Dissertation, Universität Kaiserslautern.

Maurer, F. und Pews, G. (Hrsg.). 1996. *Supporting Cooperative Work in Urban Land Use Planning*. COOP-96.

Mauss, J. 1992. *Ein heuristisch gesteuerter Chart-Parser für attributierte Graph-Grammatiken*. Dokument D-92-10. Deutsches Forschungszentrum für Künstliche Intelligenz.

Mayer, Otto. 1995. *Programmieren in COMMON LISP*. 2. Ausg. Spektrum, Akademischer Verlag.

McAllester, D. und Rosenblitt, D. 1991. Systematic Nonlinear Planning. *Seiten 634–639 in: AAAI-91*.

Muñoz-Avila, H. und Weberskirch, F. 1996. *Planning for Manufacturing Workpieces by Storing, Indexing and Replaying Planning Decisions*. To Appear: IPS-96.

Nau, D.S., Gupta, S.K. und Regli, W.C. 1995. AI Planning Versus Manufacturing-Operation Planning: A Case Study. *In: IJCAI-95*.

Öchsner, H. 1992. *Mehrdimensionale Zugriffspfadstrukturen für das ähnlichkeitsbasierte Retrieval von Fällen*. Diplomarbeit, Universität Kaiserslautern, Fachbereich Informatik.

Petrie, Ch. 1991. Context Maintenance. *Seiten 288–295 in: AAAI-91*.

Petrie, Ch. 1992. Constrained Decision Revision. *Seiten 393–400 in: AAAI-92*.

Roller, D. 1995. *Computer Aided Design*. Springer-Verlag.

Spur, Günter und Krause, Frank-Lothar. 1984. *CAD-Technik: Lehr- u. Arbeitsbuch für die Rechnerunterstützung in Konstruktion und Arbeitsplanung*. Hanser Verlag.

Stevens, Richard W. 1992. *Programmieren von UNIX-Netzen*. Prentice Hall.

Weberskirch, F. 1994. *Realisierung eines nichtlinearen Planungssystems zur Unterstützung der Arbeitsplanerstellung bei der computerintegrierten Fertigung (CIM)*. Diplomarbeit, Universität Kaiserslautern.

Weberskirch, F. 1995. *Combining SNLP-like Planning and Dependency-Maintenance*. Bericht LSA-95-10E. Centre for Learning Systems and Applications, University of Kaiserslautern, Germany.

Zeid, Ibrahim. 1991. *CAD/CAM: Theory and Practice*. McGraw-Hill, Inc.