Diplomarbeit

Erstellen eines Programmprototyps zur Erzeugung eines Manipulationsskripts für den *SinuCom Update Agent* zur Konfiguration des Schwenkzyklus CYCLE800 für Standard-Kinematiken von Kundenmaschinen

Vorgelegt am:	31.08.2009
Von:	**Peter Berg** Am Vogelherd 3B 09573 Augustusburg
Studienrichtung:	Informationstechnik
Studiengang:	Prozessinformatik
Seminargruppe:	IT 06/1
Matrikelnummer:	4060392
Praxispartner:	HEITEC AG NL Chemnitz Clemens-Winkler-Straße 3 09116 Chemnitz

Inhalt

1 Abbildungsverzeichnis

2 Abkürzungsverzeichnis

CNC	Computerized Numerical Control
HMI	Human Machine Interface
MD	Maschinendaten
NC	Numerical Control
NCU	Numerical Control Unit
OMG	Object Management Group
PCU	Personal Computer Unit
SDS	Schwenkdatensatz
UML	Unified Modeling Language
WKS	Werkstückkoordinatensystem

3 Aufgabenstellung

Diese Arbeit beschäftigt sich mit dem Schwenkzyklus CYCLE800 und dessen korrekter Inbetriebnahme. Diese Notwendigkeit besteht, da eine Vielzahl von Anfragen, dieses Thema betreffend, bei der SINUMERIK-Hotline eingehen. Die Kunden haben Schwierigkeiten, den Schwenkzyklus für ihre Zwecke richtig zu konfigurieren und in Betrieb zu nehmen. Sie erhalten daher oftmals ein unerwartetes Verhalten ihrer Maschine bzw. fehlerhafte Ergebnisse.

Die Ursachen dafür sind in der hohen Komplexität des Schwenkzyklus und dessen vielseitiger Einsetzbarkeit zu suchen. Dabei wirkt sie sich einerseits positiv auf die verschiedenen Kundenwünsche aus, ist anderseits jedoch mit einem hohen Konfigurationsaufwand und einer Reihe von möglichen Fehlerquellen behaftet.

Es ist Aufgabe dieser Arbeit, ein Hilfsmittel für den Hotline-Mitarbeiter zu entwerfen, mit dem sich die vom Kunden vorgenommenen Einstellungen überprüfen lassen. Damit kann ihm geholfen werden, seine möglicherweise fehlerhafte Konfiguration zu korrigieren.

Nach reiflicher Überlegung entstand deshalb die Idee, ein Abfrageprogramm zu entwickeln, das mit kurzen und präzisen Beschreibungen sowie Hilfebildern dem Hotline-Mitarbeiter durch eine Reihe von Eingabemasken führt. Seine Aufgabe ist es, sie mit den Angaben des Kunden bzw. dessen Maschine zu füllen.

Die vom Programm erzeugten Einstellungen sollen dabei mit denen verglichen werden, die der Kunde an seiner Steuerung gemacht hat. Bei Unterschieden der beiden Konfigurationen kann der Hotline-Mitarbeiter dem Kunden besser auf die Fehlerquellen hinweisen und ihm bei der Beseitigung helfen.

Ergebnisse dieser Arbeit sollen ein funktionsfähiges Abfrageprogramm und eine Beschreibung der Anwendung sein, die den Hotline-Mitarbeitern zur Verfügung gestellt werden. Auf die technischen Details und die benötigten Hilfsmittel wird im nächsten Kapitel näher eingegangen.

4 Analyse des Themenkomplexes

Um die Funktionalität des Abfrageprogramms schon im Vorfeld richtig abzugrenzen, soll in diesem Kapitel der Weg von der Eingabe der Daten in das Abfrageprogramm bis hin zur Auswertung und Hilfestellung beim Kunden grob skizziert werden und die auf diesem Weg benötigten Hilfsmittel sowie der technische Zusammenhang näher erläutert werden. Für das bessere Verständnis der Thematik werden als erstes der Schwenkzyklus selbst betrachtet und die für die Inbetriebnahme benötigten Einstellungen herausgearbeitet.

4.1 Der Schwenkzyklus CYCLE800

Ein Zyklus ist im Allgemeinen ein Unterprogramm, mit dessen Hilfe Bearbeitungsabläufe, die häufig auftreten bzw. speziellen Normen unterliegen, abgearbeitet werden können. Es müssen dabei nicht mehr die teilweise komplexen und umfangreichen CNC- (Computerized Numerical Control) Anweisungen, die für solche Bearbeitungen notwendig sind, eingegeben werden, sondern der Zyklus nur mit seinem Namen aufgerufen werden. Einem Zyklus müssen Versorgungsparameter übergeben werden, die die Bearbeitung an die jeweilige Situation anpassen.

Die Bearbeitung an einer CNC-Maschine findet immer in einer Bearbeitungsebene statt. In Abbildung 1 ist eine Fräsmaschine zu sehen, die die Kontur in der X-Y-Ebene bearbeitet. Die Tiefenzustellung erfolgt dabei in Richtung der Z-Koordinate. Dieser Zustand wird auch als Maschinengrundstellung bezeichnet.

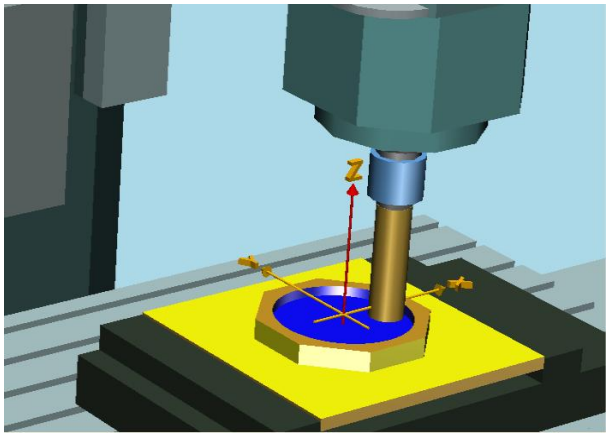

Abbildung 1: Fräsmaschine in Grundstellung [MTS_Fräs]

Der CYCLE800 ist ein spezieller Zyklus, der die Bearbeitungsebene von der Grundstellung der Maschine aus auf eine beliebige Ebene im Raum schwenkt, um die Bearbeitung dort fortsetzen zu können (siehe Abbildung 2). Dabei ist die dort zu verfahrende Kontur so zu programmieren, als würde man sich in Maschinengrundstellung befinden. Das Schwenken wird mit Hilfe von einer oder zwei Rundachsen im Werkzeugkopf und/oder Werkstücktisch realisiert und das WKS (Werkstückkoordinatensystem) mittels Schwenkframes entsprechend der Drehungen und Verschiebungen auf die neue Bearbeitungsebene angepasst. Ein Schwenkframe ist ein Satz von Nullpunktverschiebungen der Achsen, der bei der Korrektur des WKS mit den aktuellen Achspositionen verrechnet wird. Dabei wird immer gewährleistet, dass das Werkzeug senkrecht auf der Bearbeitungsebene steht.

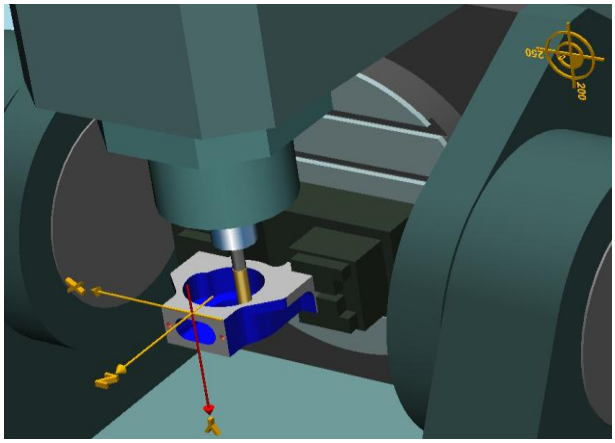

Abbildung 2: Fräsmaschine mit geschwenktem Tisch [MTS_Fräs]

Der kinematische Aufbau der Maschine kann sehr stark variieren. Der Kinematiktyp hängt einerseits von der Anzahl an Rundachsen und anderseits von der verbauten Position der Rundachsen in der Maschine ab. Rundachsen bewegen sich in Gegensatz zu Linearachsen nicht auf einer bestimmten Strecke hin und her, sondern führen eine Kreisbewegung in einem gewissen Winkelbereich aus. An einer Maschine können dadurch z.B. der Werkstücktisch gedreht oder die Orientierung des Werkzeugs verändert werden. Es sind verschiedene Aufbauformen von Maschinen möglich. Besitzt die Maschine eine Rundachse, könnte sie sich entweder am Werkzeugkopf oder im Werkstücktisch befinden. Im ersten Fall handelt es sich um den Typ Kopfkinematik und im zweiten um eine Tischkinematik. Bei zwei Rundachsen können sich beide im Werkzeugkopf, beide im Werkstücktisch oder eine im Kopf und die andere im Tisch befinden. Ersteres ist wieder eine Kopfkinematik, die zweite eine Tischkinematik. Zusätzlich gibt es noch den dritten Fall, der eine gemischte Kinematik darstellt. Der Schwenkzyklus nutzt den kinematischen Aufbau der Maschine zum Schwenken

der Bearbeitungsebene. Die Fräsmaschine in Abbildung 2 ist eine Tischkinematik mit zwei Rundachsen.

Bevor der Schwenkzyklus benutzt werden kann, muss er in Betrieb genommen werden. Dazu sind einige Einstellungen in der Konfiguration der Steuerung notwendig. Um sie vollständig zu erfassen, wurde dazu die Inbetriebnahmeanleitung des Schwenkzyklus in der Dokumentation studiert.

Es müssen eine Reihe von Maschinendaten gesetzt werden, um bestimmte Funktionalitäten freizuschalten und Speicherplatz für verschiedene Variablen zu schaffen. Weiterhin muss ein Schwenkdatensatz angelegt werden, wo Daten über den kinematischen Aufbau der Maschine, Winkelbereiche der Rundachsen sowie Schwenk- und Freifahrstrategie gespeichert werden.

4.1.1 Die Maschinendaten

MD (Maschinendaten) dienen im Allgemeinen dazu, die Steuerung an die an ihr angeschlossene Maschine anzupassen. Es können Art und Anzahl der verwendeten Achsen sowie deren Antriebsmodule, Motoren, Geber und Messsysteme definiert werden. Außerdem sind auch dynamische Werte, wie Geschwindigkeits- und Beschleunigungsverhalten der einzelnen Achsen parametrierbar. Wie man in Abbildung 3 sehen kann, besteht ein MD aus einer Nummer, einem eindeutigen Bezeichner und dem zugewiesenen Wert. Der Wert kann dabei je nach MD unterschiedlichen Datentyps sein. Es gibt in der Bedienoberfläche der Steuerung, der HMI (Human Machine Interface), einen Bedienbereich namens *Inbetriebnahme*, in dem alle vorhandenen MD aufgelistet sind. Dort können auch Veränderungen an den MD vorgenommen und die Steuerung damit projektiert werden.

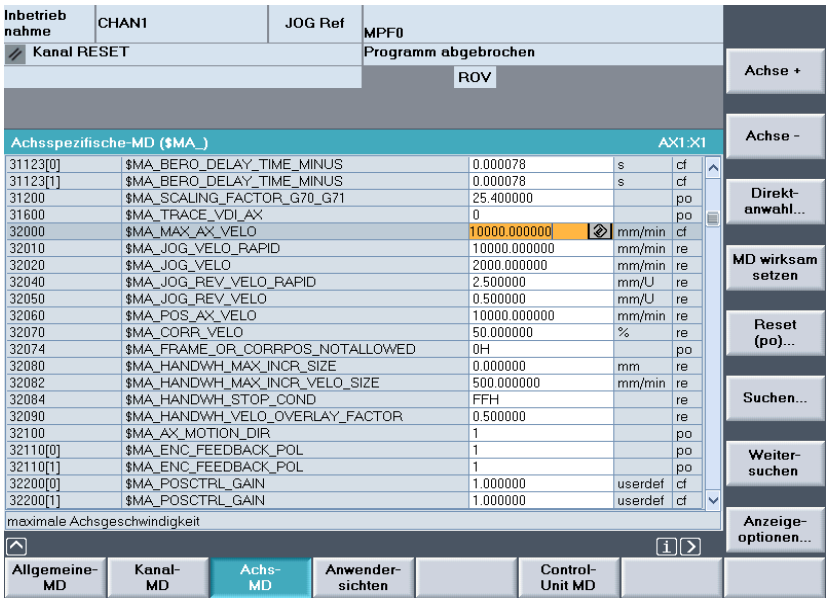

Abbildung 3: Maschinendaten [Scnsht_HMI]

Wie bereits erwähnt, dient das Setzen von MD auf einen bestimmten Wert bei der Inbetriebnahme des Schwenkzyklus dazu, die benötigten Funktionen für das Schwenken freizuschalten sowie Speicherplatz für Schwenkdatensätze und Schwenkframes zu schaffen.

In der Inbetriebnahmeanleitung des CYCLE800 befindet sich eine Tabelle mit den MD, die für den Schwenkzyklus wichtig sind. Die gesamte Tabelle ist auch im digitalen Anhang dieser Arbeit unter Tabelle 7 zu finden. Zur besseren Veranschaulichung ist ein kleiner Auszug aus dieser Tabelle in Abbildung 4 zu sehen, in der zu erkennen ist, dass jedes MD eine Wertvorgabe besitzt und die Tabelle in zwei Gruppen unterteilt ist. Dabei sind einmal MD vorhanden, die in der letzten Spalte mit einen ‘G‘ gekennzeichnet sind. Diese MD sind mit dem exakten Wert der Wertvorgabe zu versehen. Weiterhin gibt es MD, die mit einem ‘V‘ in der letzten Spalte markiert sind. Bei diesen MD ist die Vorgabe nur ein Vorschlag und der Wert kann vom Anwender in einem gewissen Rahmen frei gewählt werden.

MD-Nr.	MD-Bezeichner	Wert	Kommentar	Änderbar
10602	$MN_FRAME_GEOAX_CHANGE_MODE	1	1)	V
11450	$MN_SEARCH_RUN_MODE	Bit 1=1	Aktivieren PROG_EVENT nach Satzsuchlauf	G
11602	$MN_ASUP_START_MASK	Bit 0=1	Schwenken in JOG	V
11604	$MN_ASUP_START_PRIO_LEVEL	100	Schwenken in JOG	V
18088	$MN_MM_NUM_TOOL_CARRIER	n>0	n ⇒ Anzahl der Schwenkdatensätze1)	V
18114	$MM_ENABLE_TOOL_ORIENT	2	Werkzeuggrundorientierung	V
20108	$MC_PROG_EVENT_MASK	0	Systemasup PROG_EVENT nach Satzsuchlauf	V
20110	$MC_RESET_MODE_MASK	'H4041'	Bit 14=1	G
20112	$MC_START_MODE_MASK	'H400'	-	G
20126	$MC_TOOL_CARRIER_RESET_VALUE	0...n	wird im CYCLE800 beschrieben	V
20150	$MC_GCODE_RESET_VALUES[41]	1	TCOABS1)	G

Abbildung 4: Maschinendaten für CYCLE800-Inbetriebnahme [Doc_CYCLE800]

Diese MD sind beim Entwurf des Abfrageprogramms zu berücksichtigen und es ist darauf zu achten, dass sie alle für die jeweilige Situation den richtigen Wert erhalten.

4.1.2 Der Schwenkdatensatz

Durch den Wert eines der eben beschriebenen MD wird die maximale Anzahl der anzulegenden SDS (Schwenkdatensätze) festgelegt und Speicher dafür reserviert. In der Abbildung 5 ist zu erkennen, dass der SDS aus einer Reihe von Systemvariablen besteht, die die Werte für die zu projektierende Schwenkkinematik aufnehmen sollen. Eine komplette Auflistung aller im SDS enthaltenen Systemvariablen und ihrer Bedeutung befindet sich im digitalen Anhang in Tabelle 8.

```
CHANDATA(1)
$TC_CARR1[1]=0.175 '303e
$TC_CARR2[1]=-140.49 '30b6
$TC_CARR3[1]=-239.07 '3d96
$TC_CARR4[1]=-0.175 '32a8
$TC_CARR5[1]=140.49 '2e40
$TC_CARR6[1]=0 '2a78
$TC_CARR7[1]=0 '2b58
$TC_CARR8[1]=0 '29d8
$TC_CARR9[1]=1 '2a76
$TC_CARR10[1]=0 '2c18
$TC_CARR11[1]=1 '2d12
$TC_CARR12[1]=0 '2d48
$TC_CARR13[1]=0 '2ee8
$TC_CARR14[1]=0 '2c20
$TC_CARR15[1]=0 '2c8c
$TC_CARR16[1]=0 '2d68
$TC_CARR17[1]=239.07 '40a2
$TC_CARR18[1]=0 '2c28
```

Abbildung 5: Systemvariablen [Scnsht_ARC]

4.1.2.1 Die Systemvariablen

Systemvariablen ähneln sehr stark den MD. Sie besitzen immer einen eindeutigen Bezeichner sowie einen zugewiesenen Wert. Allerdings sind sie in der HMI-Oberfläche nicht gelistet und dienen nur zum Teil zur Konfiguration der Steuerung. Auf Systemvariablen kann nur in Teileprogramm oder Initialisierungsdateien zugegriffen werden. Es gibt Systemvariablen, die nur gelesen werden können. Diese Variablen beinhalten z.B. Positions- bzw. Drehzahlwerte von Achsen, Strom- bzw. Leistungswerte von Antrieben sowie bestimmte interne Statuswerte. In der Steuerung befinden sich aber auch Systemvariablen, die gelesen und denen auch Werte zugewiesen werden können. Dazu zählen beispielsweise die Werkzeugdaten wie die Werkzeuglänge oder der Schneidenradius und die für den Schwenkzyklus wichtigen Werkzeugträgerdaten. Außerdem können einige Systemvariablen in Echtzeit gelesen werden, was für zeitkritische Vorgänge von großer Bedeutung ist. Das bedeutet, dass von diesen Variablen in jedem Interpolationstakt der Steuerung ein sicherer Wert geliefert wird.

Wie schon beschrieben, werden bei der Inbetriebnahme des Schwenkzyklus über diese Variablen der kinematische Aufbau der Maschine wie die Abstände zwischen den Drehpunkten der Rundachsen bzw. zum Referenzpunkt, die Drehrichtungen und Winkelbereiche der Rundachsen sowie Schwenk- und Freifahrstrategie festgelegt. Sie können dann von der NCU (Numerical Control Unit) zur Berechnung der Drehung und Verschiebung des WKS und zur mechanischen Bewegung der Rundachsen verwendet werden. Die Bezeichnung für diese Systemvariablen lautet *$TC_CARRm[n]*, wobei *m* für die Nummer der Systemvariablen und *n* für die Nummer des SDS steht.

4.1.2.2 Anlegen von Schwenkdatensätzen

Das Anlegen der SDS erfolgt beim Kunden in der HMI über die Inbetriebnahmemasken des CYCLE800. Man hat die Möglichkeit, jeden SDS über drei solcher Masken zu editieren. Abbildung 6 zeigt die erste Maske mit dem Namen *Kinematik*, in der man den Namen des SDS, den Typ der Kinematik und die kinematischen Abmessungen der Maschine angeben kann. Außerdem lassen sich mögliche Schwenk- und Freifahrstrategien vorwählen, die beim Schwenken benutzt werden.

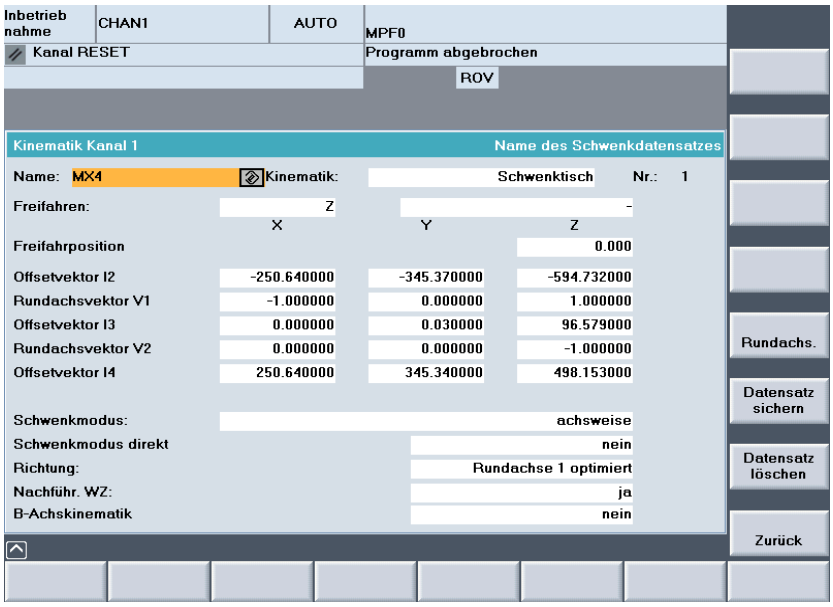

Abbildung 6: Inbetriebnahme Schwenkdatensatz - Kinematik [Scnsht_HMI]

Die Abbildung 7 zeigt die zweite Maske mit der Bezeichnung *Rundachsen*. Sie ist für die Konfiguration der am Schwenken beteiligten Rundachsen zuständig. Dabei können Achsbezeichnungen, Achsmodus, Winkelbereiche und Offsetwerte der Rundachsen festgelegt werden.

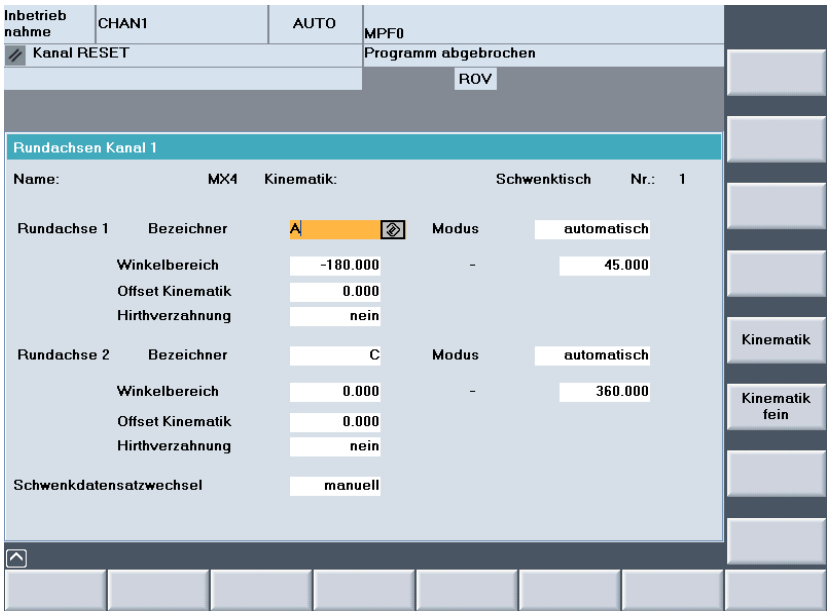

Abbildung 7: Inbetriebnahme Schwenkdatensatz - Rundachsen [Scnsht_HMI]

Die dritte Maske mit dem Namen *Kinematik fein* ist in Abbildung 8 zu sehen. Hier kann man zusätzlich zu den Abmessungswerten der Maschine und den Offsetwerten der Rundachsen noch Feinverschiebungen definieren. Die Angabe der Werte in dieser Maske ist optional.

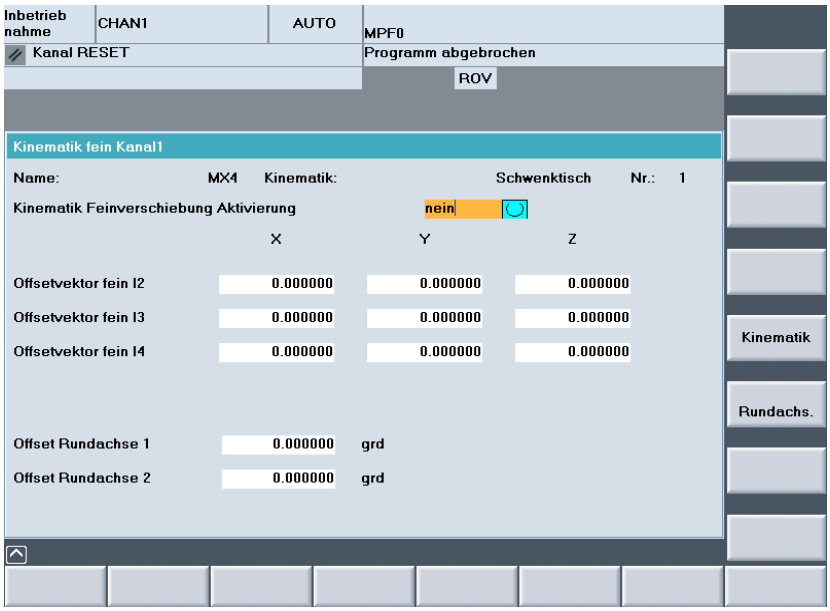

Abbildung 8: Inbetriebnahme Schwenkdatensatz - Kinematik fein [Scnsht_HMI]

Durch das Ausfüllen der Inbetriebnahmemasken werden im Hintergrund die einzelnen Systemvariablen des jeweiligen SDS mit Werten versorgt und somit die Schwenkkinematiken in Betrieb genommen.

Der CYCLE800 kann danach mittels Aufruf und entsprechenden Übergabeparametern im Teileprogramm verwendet werden. Abbildung 9 zeigt die Eingabemaske der Zyklusunterstützung, die die Versorgungsparameter im Zyklusaufruf richtig platzieren hilft.

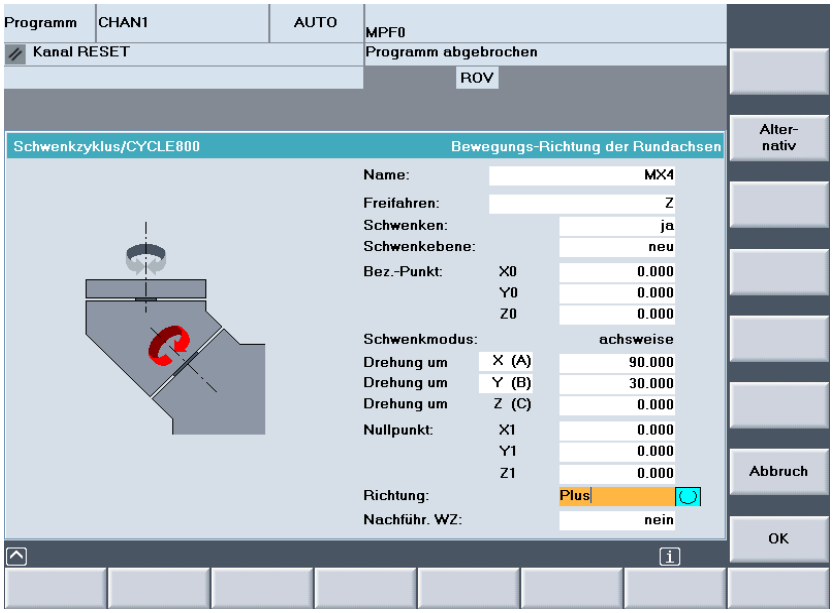

Abbildung 9: Zyklenunterstützung CYCLE800 [Scnsht_HMI]

4.2 Ablauf des Kontrollvorgangs

Da nun die theoretischen Grundlagen für das Verständnis und die Inbetriebnahme des Schwenkzyklus abgehandelt wurden, soll dieser Abschnitt den Weg von der Eingabe in das zu entwerfende Abfrageprogramm bis hin zur Hilfestellung beim Kunden skizzieren und die dabei benötigten Hilfsmittel erläutern.

4.2.1 Das Abfrageprogramm

Das Abfrageprogramm, was Aufgabe dieser Arbeit ist, muss im Wesentlichen das gleiche leisten, was auch die Inbetriebnahmemasken in der HMI erbringen. Allerdings soll die Eingabe der einzelnen Parameter durch zusätzliche Beschreibungen und Abbildungen erleichtert und dadurch mögliche Falscheingaben oder Fehlinterpretationen vermieden werden.

Der Hotline-Mitarbeiter wird während des Abfrageprogramms durch eine Reihe von Informations- und Eingabeseiten geführt. Dabei soll er Hinweise über die korrekte Inbetriebnahme des Schwenkzyklus erhalten. Präzise Erklärungen und aussagekräftige Hilfebilder sorgen dafür, dass die einzugebenden Parameter richtig interpretiert und ausgefüllt werden können. Der Hotline-Mitarbeiter muss gegebenenfalls Rücksprache mit dem Kunden nehmen, um noch fehlende Information über dessen Maschine einzuholen und sich des korrekten kinematischen Aufbaus zu versichern.

Weiterhin ist es wichtig zu wissen, was das Programm nach der vollständigen Eingabe aller Parameter als Ergebnis liefern soll. Die Inbetriebnahme des Schwenkzyklus beim Kunden selbst ist nicht möglich, da dem Hotline-Mitarbeiter weder die Maschine des Kunden noch dessen Steuerung zur Verfügung steht. Es ist allerdings möglich, die nötigen Einstellungen an einem Serieninbetriebnahme-Archiv vorzunehmen.

4.2.2 Das Serieninbetriebnahme-Archiv

Ein Serieninbetriebnahme-Archiv ist eine Datensicherung der gesamten Steuerung. Es können dabei der aktuelle Zustand aller MD, die geladenen Teileprogramme, Antriebsdaten sowie die verwendeten Systemvariablen wie Werkzeug- und Werkzeugträgerdaten gesichert werden. Wie in Abbildung 10 ersichtlich ist, gibt es in der HMI im Bedienbereich *Dienste* die Möglichkeit, ein solches Serieninbetriebnahme-Archiv zu erstellen.

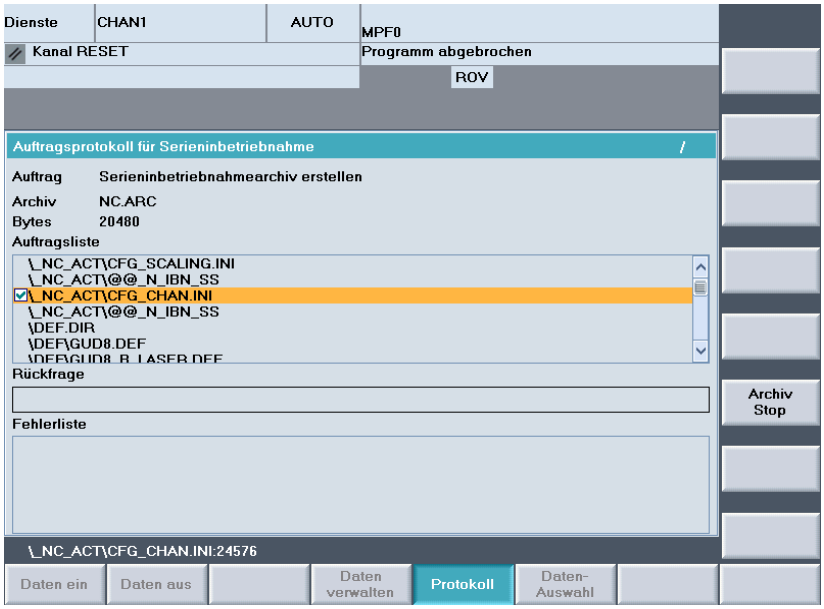

Abbildung 10: Serieninbetriebnahme erstellen [Scnsht_HMI]

Abbildung 11 zeigt, dass das Archiv selbst eine teilweise binär kodierte Textdatei ist, bei der die MD, Systemvariablen und die Teileprogramme zeilenweise untereinander angeordnet sind.

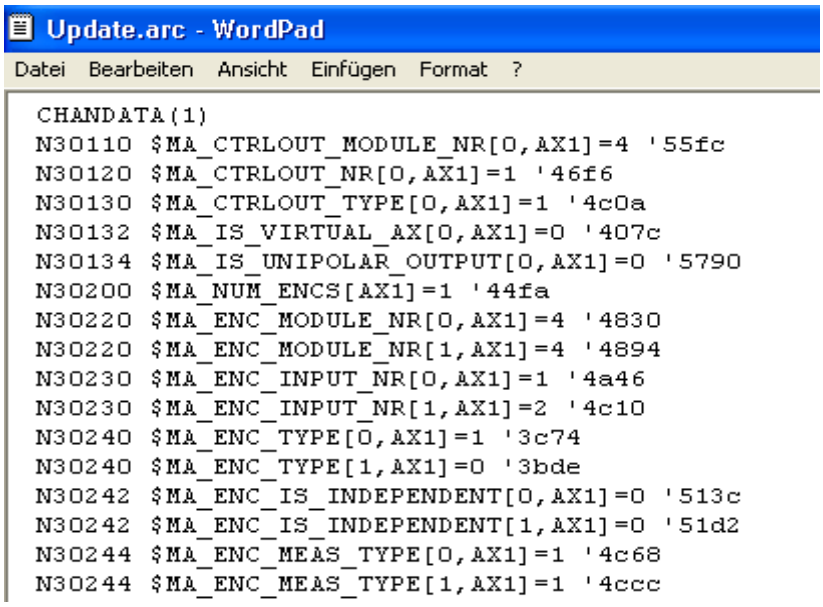

Abbildung 11: Serieninbetriebnahme-Archiv [Scnsht_ARC]

Es ist erforderlich, dass der Kunde an seiner Steuerung ein Serieninbetriebnahme-Archiv erzeugt und es dann der SINUMERIK-Hotline zur Untersuchung zusendet. Zum Vergleich stellt man gleichfalls ein Archiv mit den aus dem Abfrageprogramm gewonnenen Einstellungen her. Um dabei ungewünschte Seiteneffekte durch andere Veränderungen im Archiv zu vermeiden, ist es ratsam, eine Kopie des Kundenarchivs zu verwenden und die nötigen Veränderungen dort vorzunehmen. Gib es dann Unterschiede zum Original-Kundenarchiv, sind die für den Schwenkzyklus wichtigen MD bzw. Systemvariablen gefunden. Der Grund des Unterschieds kann untersucht werden, um dem Kunde damit helfen zu können.

Das Abfrageprogramm kann allerdings diese Einstellungen nicht selbst vollziehen. Hinter den MD und den Systemvariablen befindet sich eine Zeilenprüfsumme, die bei Veränderung des Wertes angepasst werden müsste. Da der Algorithmus für die Berechnung der Prüfsummen von Siemens geheim gehalten wird, ist es nicht möglich, die Veränderungen am Archiv direkt vorzunehmen. Desweiteren sind die Einstellungen auf binärer Ebene durchzuführen, was in der kurzen Entwicklungszeit dieses Programms einfach nicht zu realisieren war.

Es existiert jedoch ein Programm, mit dessen Hilfe man Manipulationen an Serieninbetriebnahme-Archiven vornehmen kann und die Berechnung der Prüfsumme bei Veränderung des Wertes automatisch stattfindet. Dieses Programm ist der *SinuCom Update Agent*.

4.2.3 SinuCom Update Agent

Der *SinuCom Update Agent* ist ein Software-Paket, mit dessen Hilfe Anpassungen und Hochrüstungen von Steuerungen an Serienmaschinen vorgenommen werden können. Dabei wird ein Serieninbetriebnahme-Archiv von einer Referenzmaschine erzeugt. Danach können Veränderungen des Archivs und Dateien für Software-Hochrüstungen in einem Update-Paket projektiert und anschließend an einer baugleichen Zielmaschine die Veränderungen vorgenommen werden. Diese Vorgehensweise ist sehr effektiv, wenn man eine Reihe von Maschinen in kürzester Zeit in Betrieb nehmen muss.

Der Update Agent besteht aus den vier Komponenten: *UPExpert*, *UPShield*, *UPDiff* und *UPTopo*. Es soll hier nicht jede einzelne Funktion dieser Komponenten erläutert werden, sondern nun der für diese Arbeit erforderliche Weg geschildert und die dafür benötigten Vorbereitungen getroffen werden. Es werden dazu die ersten drei Komponenten benötigt. *UPTopo* ist für das Projektieren von Antriebsobjekten notwendig, hat aber für diese Arbeit keine Bedeutung.

4.2.3.1 UPExpert

UPExpert wird für das Erstellen der Update-Pakete und das Einbindung des Quellarchivs verwendet.

Nachdem das Programm gestartet wurde, ist zuerst ein neues Paket anzulegen. Das erreicht man durch das Auswählen des Menüpunktes *Neu…* im Hauptmenü unter *Datei*. Dabei öffnet sich ein Fenster wie in Abbildung 12. Es ist der Pakettyp *Archive* einzustellen, da ausschließlich nur auf einem Serieninbetriebnahme-Archiv gearbeitet werden soll. Zusätzlich sind noch ein Paketname und der Ablageort anzugeben.

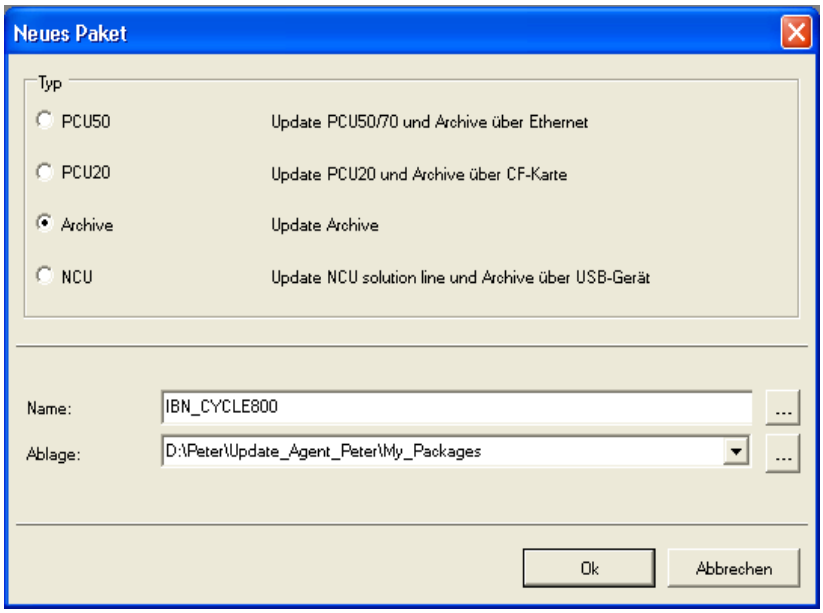

Abbildung 12: Neues Paket anlegen in UPExpert [Scnsht_UA]

Nach Betätigen des *OK*-Buttons findet man die Oberfläche wie in Abbildung 13 vor. Auf der linken Seite des Bildschirms befinden sich die fünf Reiterkarten: *Paket*, *Dialoge*, *Topologie*, *Schritte* und *C-Navigator*, wobei für diese Arbeit nur die erste und die beiden letzten Reiterkarten bearbeitet werden müssen.

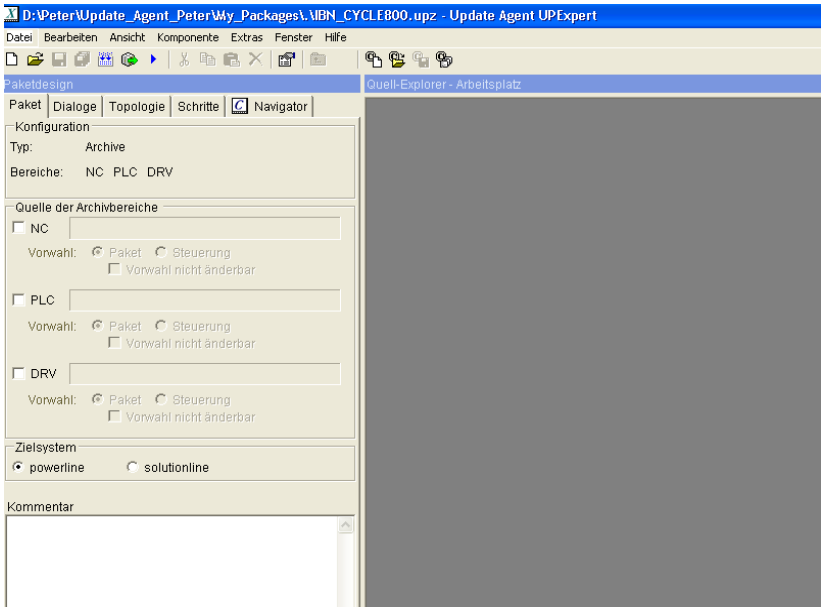

Abbildung 13: Oberfläche von UPExpert [Scnsht_UA]

Auf der Reiterkarte *Paket* ist bei *Quelle des Archivbereichs* der Bereich *NC* zu wählen und bei *Vorwahl* der Punkt bei *Paket* zu setzen. Der Haken bei *Vorwahl nicht änderbar* sollte entfernt bleiben. Neben dem Bereich *NC* befindet sich ein leeres Feld und rechts daneben ein Button

mit einem kleinen Pfeil nach rechts. Um jetzt das zu verändernde Archiv, also die Kopie des Kundenarchivs, in das Paket einzubinden, betätigt man diesen Button und wählt im Untermenü den Punkt: *in Paket einfügen…*. Jetzt sucht man im Dateibrowser das zu verändernde Archiv heraus und bestätigt mit *Öffnen*. Als letzte Einstellung auf dieser Reiterkarte ist das Zielsystem zu wählen, auf dem das veränderte Archiv am Ende getestet werden soll. Für diese Aufgabe wird das Zielsystem *powerline* gewählt. Die Reiterkarte könnte dann wie in Abbildung 14 aussehen.

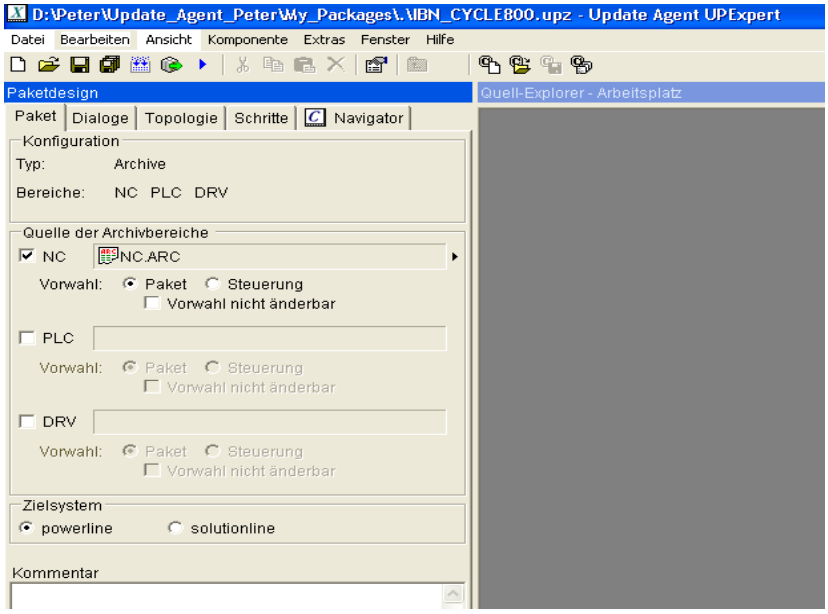

Abbildung 14: Paketdesign - Paket in UPExpert [Scnsht_UA]

Nachfolgend ist die Reiterkarte *Schritte* zu aktivieren. Hier kann man eine Reihe von Schritten in Form eines Schrittbaumes anlegen, um den Veränderungen am Archiv eine Struktur zu verleihen. Für diese Arbeit wird nur ein Schritt benötigt. Es ist zu Beginn schon ein Schritt mit dem Namen *Neuer Schritt* vorhanden, der noch über Betätigen der rechten Maustaste und Auswählen des Untermenüpunkts *Umbenennen* eine aussagekräftige Bezeichnung erhalten muss. Hierfür ist der Namen *CYCLE800* gewählt worden.

Für diesen Schritt ist jetzt eine Komponente zu erzeugen und ihm unterzuordnen. Eine Komponente ist eine Art Behälter für Änderungs- und Manipulationsaufträge. Das Anlegen erfolgt, in dem man mit der rechten Maustaste auf den Schritt klickt und im Untermenü den Punkt *Neue Komponente verknüpfen…* wählt. Wenn sich der Komponenteneditor geöffnet hat, wählt man oben links das gelbe *C* für interne Komponente, gibt ihr unten einen Namen, z.B.: *CP_CYCLE800*, und bestätigt mit *Erstellen*. Als letztes auf dieser Reiterkarte muss im

Kästchen links neben dem Schrittnamen der grüne Haken gesetzt werden. In Abbildung 15 sind die getätigten Einstellungen zu sehen.

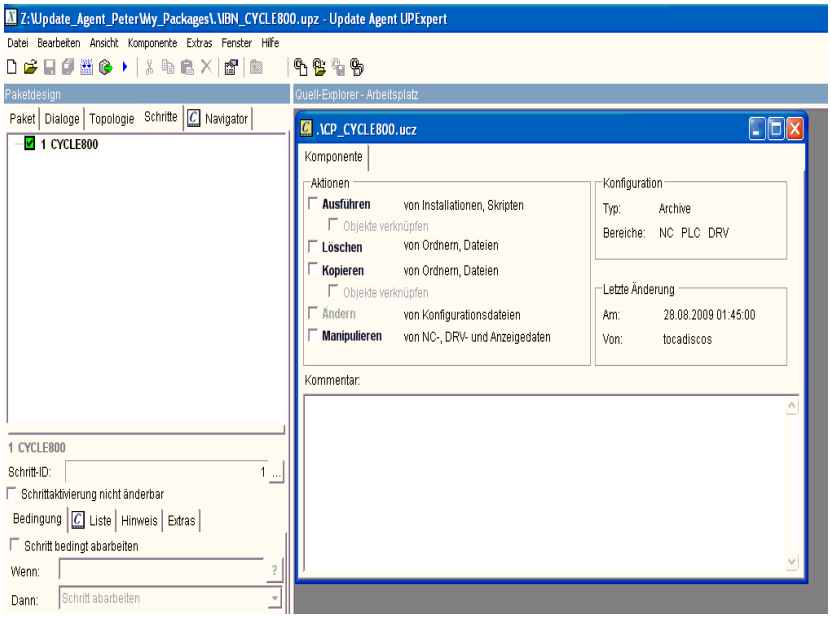

Abbildung 15: Paketdesign - Schritte in UPExpert [Scnsht_UA]

Nachdem die letzte Reiterkarte markiert wurde, muss im Komponentenfenster rechts daneben das Kästchen bei *Manipulieren* angehakt werden. Es erscheint ein neuer Reiter oben mit dem Namen *Manipulieren*. Nach dessen Auswahl, unterteilt sich die Anzeige des Komponentenfensters in zwei Sektionen mit den Überschriften *Zielbereich* und *Anweisungen*. Hier ist jetzt ein Manipulationsauftrag anzulegen und einem Zielbereich unterzuordnen. Da für diese Arbeit ausschließlich NC (Numerical Control)-Daten manipuliert werden sollen, ist auf diesen Zielbereich mit der rechten Maustaste zu klicken und über das Untermenü ein neuer Auftrag anzulegen.

Jetzt gibt es zwei Möglichkeiten. Entweder man legt über den Untermenüpunkt *Neuer Auftrag* einen leeren Auftrag an und füllt ihn in der rechten Fensterhälfte mit Anweisungen oder bindet über den Untermenüpunkt *Auftrag verknüpfen* eine externe Skriptdatei ein.

An dieser Stelle bietet sich die Möglichkeit, eine Schnittstelle zwischen Abfrageprogramm und Update Agent zu schaffen. Wenn das Abfrageprogramm eine solche Skriptdatei mit den entsprechenden Manipulationsanweisungen erzeugt, kann der Update Agent mit Hilfe dieser Skriptdatei das Archiv mit den gewünschten Einstellungen manipulieren. Man bindet also

nach Erzeugung der Skriptdatei durch das Abfrageprogramm diese über Untermenüpunkt *Auftrag verknüpfen* als Manipulationsauftrag in den Update Agent ein. Abbildung 16 zeigt die Reiterkarte nach diesem Vorgang.

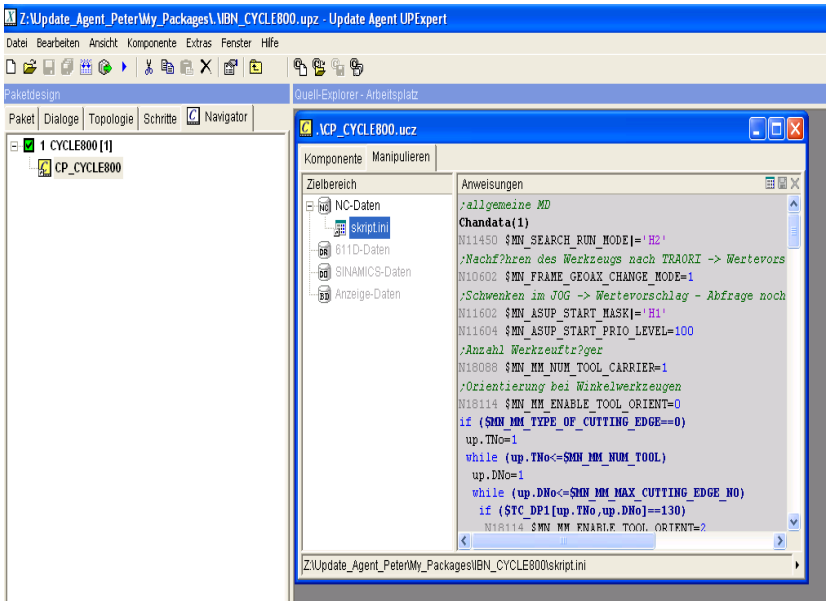

Abbildung 16: Paketdesign - C-Navigator in UPExpert [Scnsht_UA]

Über den Menüpunkt *Speichern* im Hauptmenü *Datei*, ist das gesamte Paket zu speichern.

Die letzte Aufgabe für *UPExpert* ist die Abarbeitung des Pakets mit der Softwarekomponente *UPShield* vorzubereiten. Der Vorgang *Paketweitergabe* wird über den Menüpunkt *Weitergeben mit UPShield…* im Hauptmenü unter *Datei* gestartet. Dabei ist zuerst der Ablageort für das weitergegebene Paket im Dateibrowser auszuwählen und mit *OK* zu bestätigen. Danach folgt ein Überprüfungslauf, der die Integrität des Pakets überprüft. Wenn er ohne Beanstandung durchlaufen wurde, beginnt die eigentliche Paketweitergabe, wie in Abbildung 17 zu sehen ist. Als Ergebnis dieses Vorgangs entsteht im angegebenen Ablageort eine ausführbare Anwendung (exe-Datei), die den gleichen Name wie das Paket trägt.

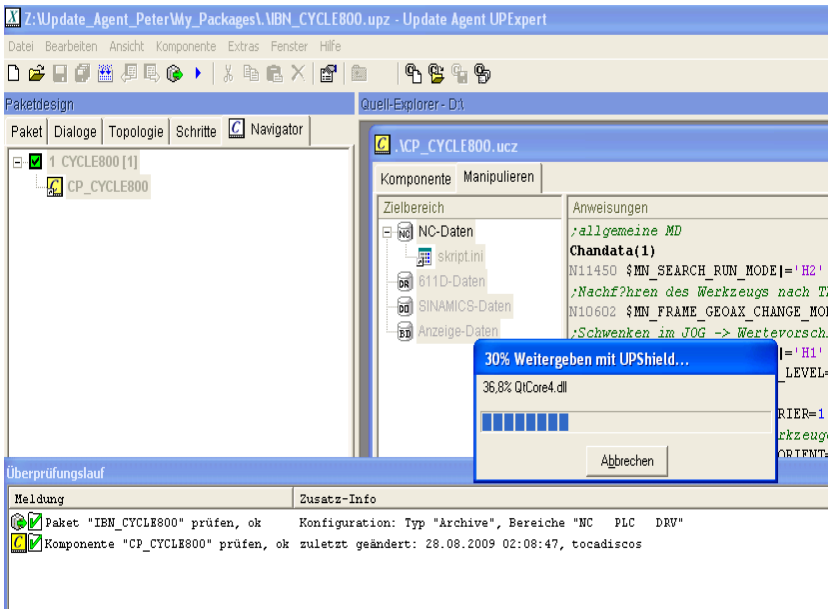

Abbildung 17: Paketweitergabe in UPExpert [Scnsht_UA]

4.2.3.2 UPShield

Diese Softwarekomponente dient zur Abarbeitung des Pakets und zum Ausführen der projektierten Veränderungen. Dazu muss zuerst die mit *UPExpert* erzeugte Anwendung auf die PCU (Personal Computer Unit) der Steuerung, auf der das veränderte Archiv getestet werden soll, kopiert und dort ausgeführt werden. Dabei startet *UPShield* und führt den Anwender durch eine Reihe von Dialogseiten, die man nacheinander bearbeiten muss.

Auf der ersten Seite ist als auszuführendes Updatepaket die erzeugte exe-Datei und als Menüsprache *Deutsch* auszuwählen. Mit *Weiter* gelangt man jeweils auf die nächste Seite. Der folgende Dialog dient zur Angabe des Zugriffswegs der Steuerung. Dabei muss der Computername der PCU angegeben werden. Wenn die Anwendung auf der PCU ausgeführt wird, trägt UPShield den Namen selbst ein. Auf Seite drei ist auszuwählen, ob das Archiv aus dem Paket oder das der angeschlossenen Steuerung bearbeitet werden soll. Für diese Arbeit ist das Archiv des Pakets zu wählen. Im nächsten Dialog sind die projektierten Schritte anzuhaken, die abgearbeitet werden sollen. Durch die Auswahl der Schritte können einzelne Steuerungen mit unterschiedlichen Veränderungen über das gleiche Paket versehen werden. Hier muss der eine projektierte Schritt angehakt bleiben. Nach Betätigen des *Fertig*-Buttons erscheint die Seite für die Schrittabarbeitung, in dem der Abarbeitungstyp *Durchlauf* zu wählen ist. Mit einem Mausklick auf *START* wird der projektierte Manipulationsauftrag

ausgeführt. In Abbildung 18 ist dieser Vorgang zu sehen. Als Ergebnis wird ein verändertes Archiv mit dem Namen *UPDATE.ARC* erzeugt und in das Archivverzeichnis der Steuerung kopiert.

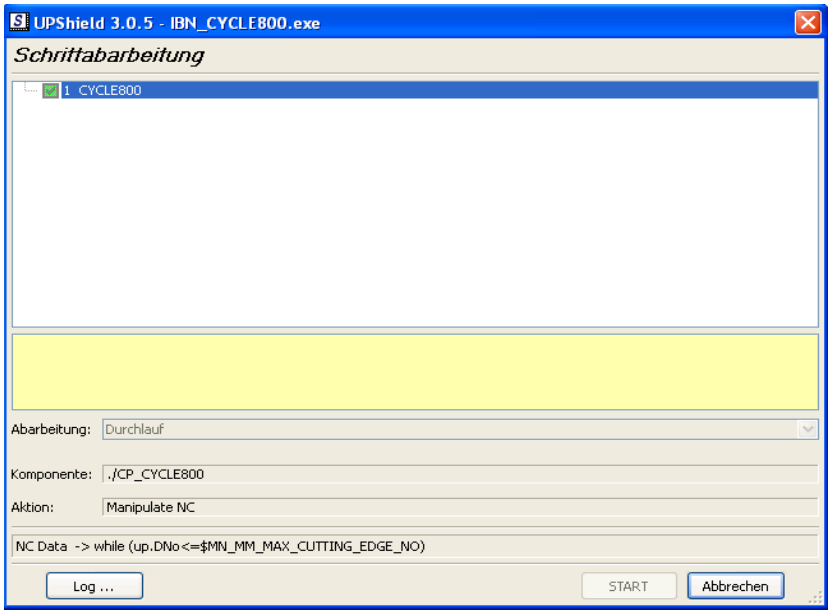

Abbildung 18: Paketabarbeitung in UPShield [Scnsht_UA]

Dieses Archiv kann nun in die Steuerung eingelesen und getestet oder auf den Arbeitsrechner kopiert und mit dem Original-Kundenarchiv verglichen werden.

4.2.3.3 UPDiff

Für den Vergleich zweier Archive wird die Softwarekomponente *UPDiff* verwendet. Nach dem Start des Programms muss zuerst der Vergleichstyp gewählt werden. Dazu findet man in der Menüleiste oben links drei Symbole: Ordnervergleich, Dateivergleich und NC-Datenvergleich. Für diese Arbeit ist der NC-Datenvergleich geeignet. Als nächstes sind die beiden Archive in den Vergleich einzubinden. Rechts neben dem oberen freien Feld befindet sich ein kleines Ordnersymbol. Durch Klicken auf dieses Symbol kann man über den Dateibrowser die beiden zu vergleichen Archive nacheinander einfügen. Zur besseren Übersichtlichkeit bieten sich an, zuerst das Original-Kundenarchiv und danach das veränderte Archiv auszuwählen. Um den Vergleichsfilter einzustellen, sind oben in der Menüleiste die Symbole: Pfeil nach rechts, Ungleichheit, Gleichheit und Pfeil nach links entsprechend zu markieren. Das Ungleichheit-Symbol sollte als einziges aktiviert sein, um alle für diese Arbeit

unwichtigen Vergleichsergebnisse auszuschließen. Außerdem ist darauf zu achten, dass links die Reiterkarte für den Vergleichsmodus Normal angewählt ist. Diese Einstellungen können in Abbildung 19 nachvollzogen werden.

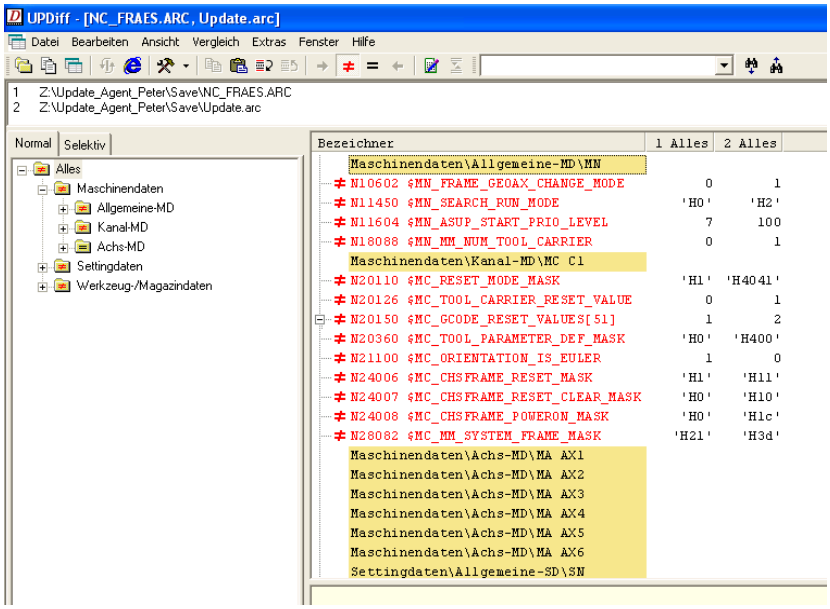

Abbildung 19: Archivvergleich in UPDiff [Scnsht_UA]

In der Ergebnisliste sind jetzt alle Daten zu sehen, die in den beiden Archiven unterschiedlich sind. Dabei steht in der Liste in der linken Spalte der Bezeichner des Datums, in der mittleren Spalte der Wert im Original-Kundenarchiv und in der rechten Spalte der Wert im veränderten Archiv. Durch Markieren der jeweiligen Zeile wird im unteren Feld zu jedem Datum eine kleine Information angezeigt.

Mit der Analyse dieser Daten kann dem Kunden schließlich die entsprechende Hilfestellung zur Lösung seines Problems gegeben werden.

5 Spezifikationen für den Programmentwurf

Nachdem der Ablauf des Kontrollvorgangs erläutert und die Aufgaben des Abfrageprogramms sowie dessen Schnittstelle abgegrenzt wurden, sind in diesem Kapitel einige Festlegungen für Laufzeit- und Programmierumgebung zu treffen. Desweiteren sollen Grundlagen für die Modellbildung und Darstellung des Programms gelegt werden, die man in der Entwurfsphase benötigt.

5.1 Wahl der Laufzeitumgebung

Prinzipiell kann das Abfrageprogramm auf einer beliebigen Windowsumgebung laufen, die Fensterdarstellung mittels .NET-Framework unterstützt. Allerdings ist es ratsam, sich für die Wahl von Windows XP zu entscheiden, da der *SinuCom Update Agent* nur unter diesem Betriebssystem lauffähig ist und die Skriptdatei vom Abfrageprogramm an den Update Agent übergeben werden muss. Ein anderes Betriebsystem kommt nur in Frage, wenn zusätzlich eine virtuelle Maschine mit Windows XP verwendet wird, auf der der Update Agent dann lauffähig ist.

5.2 Wahl der Programmierumgebung

Bei der Wahl der Programmierumgebung wird *Visual C#* des *Microsoft Visual Studios* favorisiert, da einerseits damit Fenster mit vielfältigen Eingabe- und Anzeigeelementen und anderseits mittels einer entsprechenden Dateiarbeitsklasse die für den Update Agent benötigte Skriptdatei erzeugt werden kann.

5.3 Grundlagen zur Modellierung

Da in dieser Arbeit ein Softwareprodukt entwickelt werden soll, ist es notwendig, in der Entwurfsphase ein Modell zu erstellen und dieses mit verschieden Darstellungsarten zu visualisieren. Eine derzeit weit verbreitete Entwurfsmethode ist der objektorientierte Ansatz

mittels UML (Unified Modeling Language). Da sie für die Entwicklung des Abfrageprogramms verwendet werden soll, wird in diesem Abschnitt ein Einblick in dieses Thema gegeben und die für den Entwurf benötigten Darstellungsformen herausgearbeitet und erläutert.

Die Grundlagen dieses Themas wurden mit dem Wissen aus der Vorlesung Softwareentwicklung von Professor Munke und dessen Vorlesungsskript erarbeitet. Zur Erstellung des Modells und der Diagramme ist die Modellierungssoftware ObjectiF der Firma microTOOL verwendet wurden.

5.3.1 Unified Modeling Language

Die UML ist eine standardisierte Sprache für das Erzeugen und Visualisieren von Modellen für die Entwicklung von Softwaresystemen sowie Produkte anderer Bereiche. Durch die standardisierte Sprachnotation können Entwickler auf einer einheitlichen Basis ihre Ideen austauschen. UML wurde von den “drei Amigos“ ins Leben gerufen, die ihre selbst entwickelten Entwurfsmodelle zu einem verschmelzen und damit einen gemeinsamen Standard für die Allgemeinheit schaffen wollten. Die offizielle Standardisierung wird seit 1998 von der Object Management Group (OMG) verwaltet und bis zum heutigen Stand UML 2.0 weiterentwickelt.

Die Notation der UML definiert bestimme Sprachelemente, die man als Dinge bezeichnet sowie Beziehungen zwischen den Dingen. Grafische Festlegungen für die einzelnen Dinge und Beziehungen ermöglichen es, sie in Diagrammen darzustellen. Ein einzelnes Diagramm kann allerdings niemals das komplette System beschreiben, sondern immer nur die Sicht auf einen Teil des Gesamtsystems geben. Aus diesem Grund gibt es mehrere verschiedene Diagramme, die jeweils auch nur einen Teil aller Dinge und Beziehungen zeigen können. Welche Dinge und Beziehungen visualisiert werden, hängt vom Diagrammtyp ab. In Abbildung 20 sieht man eine Aufstellung aller dreizehn in UML 2.0 bekannten Diagramme, die wiederum in sechs Strukturdiagramme und sieben Verhaltensdiagramme unterteilt sind. Strukturdiagramm stellen strukturelle Zusammenhänge zwischen Dingen dar. Verhaltensdiagramme beschreiben dagegen zeitliche oder ereignisgesteuerte Verläufe.

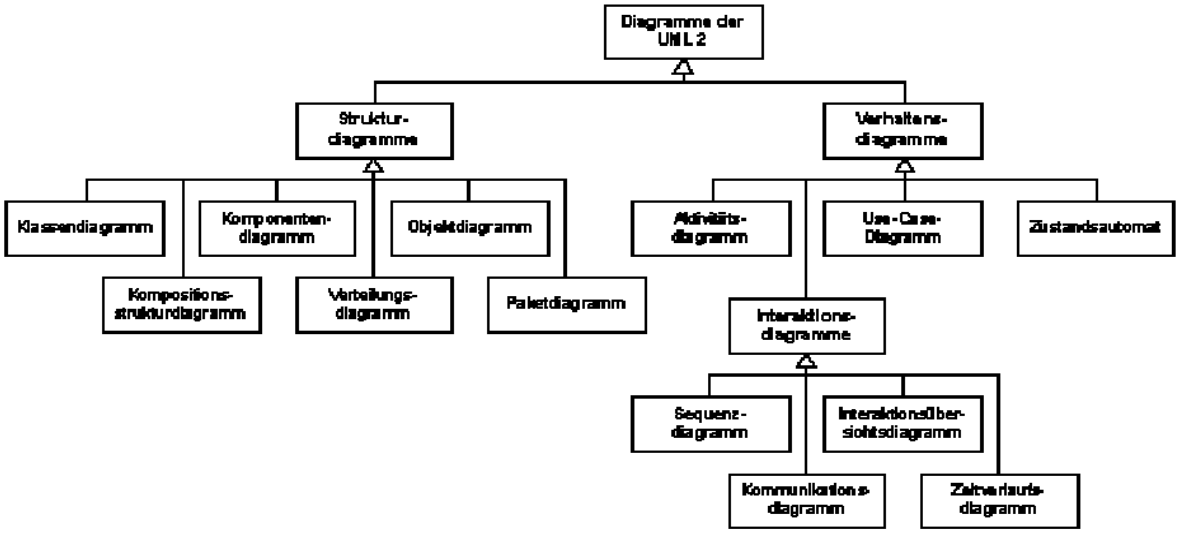

Abbildung 20: Diagramme UML2.0 [Script_SE]

Es soll hier nicht jedes dieser Diagramme beschrieben werden, sondern nur ein Einblick in den Aufbau sowie die darin vorkommenden Dinge und Beziehungen der Diagramme gegeben werden, die für diese Arbeit verwendet werden sollen. Es handelt sich dabei um das Klassen-, das Anwendungsfall-, das Aktivitäts- und das Zustandsdiagramm.

5.3.1.1 Das Klassendiagramm

Da die Entwurfsmethode auf der Objektorientierung basiert, spielen in diesem Diagramm die Klassen auch die zentrale Rolle und stellen damit die Dinge für diesen Diagrammtyp dar. Es können ebenfalls Objekte, also Instanzen dieser Klassen verwendet werden, um z.B. aktuelle Zustände von Klassenelementen zeigen zu können. Falls sich allerdings nur Objekte in einem solchen Diagramm befinden, handelt es sich um ein Objektdiagramm. Wenn es für den Darstellungskontext notwendig ist, können auch Mitgliedsvariablen und -methoden sowie deren Sichtbarkeit angegeben werden. Das Klassendiagramm gehört zur Gruppe der Strukturdiagramme und beschreibt mit Aufbau und Beziehungen strukturelle Zusammenhänge zwischen den Klassen.

Eine Klasse wird im Diagramm als Rechteck mit dem Klassennamen als Inhalt dargestellt, wobei der Name mit einem Großbuchstaben beginnen sollte. Die Mitgliedsvariablen und -methoden werden unter dem Klassenamen mit einem Querstrich von ihm getrennt platziert.

Sie sind auch voneinander mit einem Querstrich zu separieren und ihre Namen sollten mit einem Kleinbuchstaben beginnen. Die Methoden werden als kompletter Funktionsprototyp mit Übergabe- und Rückgabetypen geschrieben. Die Sichtbarkeit der Variablen und Methoden kennzeichnet man mit einem vorangestellten Symbol, wie z.B. ‘+‘ für *public*, ‘#‘ für *protected* oder ‘-‘ für *private*.

Es gibt im Allgemeinen zwei Typen von Beziehungen zwischen den Klassen. Zum einen ist das die Generalisierung, die zwischen einer generellen und einer spezielleren Klasse besteht. Die speziellere Klasse besitzt alle Merkmale der generellen Klasse, ohne sie extra noch einmal definieren zu müssen, kann aber noch zusätzliche Merkmale aufweisen, die die generelle Klasse nicht hat. Man sagt auch, die speziellere Klasse “erbt“ von der generellen Klasse. Dargestellt wird diese Beziehung mit einer durchgezogenen Linie zwischen zwei Klassen, wobei sich am Ende der generellen Klasse eine nicht gefüllte Pfeilspitze befindet.

Die zweite Beziehung ist die Assoziation. Sie ist die allgemeinste Beziehung und besteht dann, wenn zwei Klassen auf irgendeiner Art und Weise miteinander zu tun haben. Eine durchgezogene Linie zeigt diese Beziehung an. Durch die Angabe von Kardinalitätszahlen an den Enden der Linie kann der Grad der beteiligten Klassen angegeben werden.

Eine Spezialform der Assoziationsbeziehung sind Aggregation und Komposition. Sie beschreiben eine Teil-des-Ganzen-Beziehung zwischen Klassen. Bei der Komposition kommt noch gegenüber der Aggregation hinzu, dass die Teil-Klasse nicht ohne die Ganzen-Klasse existieren kann. Die Aggregation wird mit einer unausgefüllten Raute und die Komposition mit einer ausgefüllten Raute am Ende der Linie der Ganzen-Klasse dargestellt. Zur Verdeutlichung der genannten Begriffe zeigt Abbildung 21 ein einfaches Beispiel eines Klassendiagramms für eine Küche.

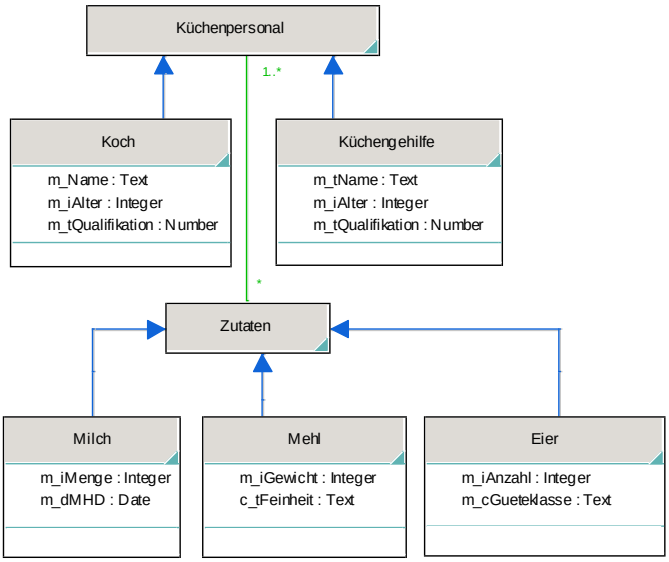

Abbildung 21: Klassendiagramm - Küche [ObjectiF]

5.3.1.2 Das Anwendungsfalldiagramm

Dieser Diagrammtyp ist ein Verhaltensdiagramm und beschreibt die Sicht des Anwenders von außen auf das System. Es wird meist für die Spezifikation der Anforderungen an das System und denen vom Benutzer erwarteten Funktionen verwendet.

Die Dinge in diesem Diagramm sind zum einen Akteure, welche den Einfluss von außen auf das System verdeutlichen sollen und zum anderen die Anwendungsfälle, die die Funktionen beschreiben, die von den Akteuren beeinflusst oder nur beobachtet werden können. Ein Anwendungsfall ist nicht als eine einzelne Aktion zu sehen, die das System ausführen soll. Vielmehr kann er mit einer Reihe von Aktivitäten versehen werden und stellt damit eine Zusammenfassung aller für eine bestimmte Funktion benötigten Aktivitäten dar. Deshalb werden die einzelnen Anwendungsfälle meist mit Aktivitätsdiagrammen verfeinert und dadurch die einzelnen Aktivitäten näher erläutert.

Ein Akteur wird im Diagramm durch ein Strichmännchen dargestellt. Für die Anwendungsfälle zeichnet man Ellipsen, die den Namen des jeweiligen Anwendungsfalls oder eine eindeutige Nummer tragen.

Es können Beziehungen zwischen Anwendungsfällen und Akteuren sowie zwischen den Anwendungsfällen untereinander bestehen, die mit einer durchgezogenen Linie miteinander verbunden werden. Es gibt auch die Möglichkeit von Generalisierungsbeziehungen zwischen den Akteuren, wenn das für die Darstellung benötigt wird. Sie wird wie beim Klassendiagramm durch eine Linie mit Pfeil an der Seite des generellen Akteurs veranschaulicht. In Abbildung 22 ist ein einfaches Anwendungsfalldiagramm, das die Aufgaben eines Kochs in seiner Küche beschreiben soll.

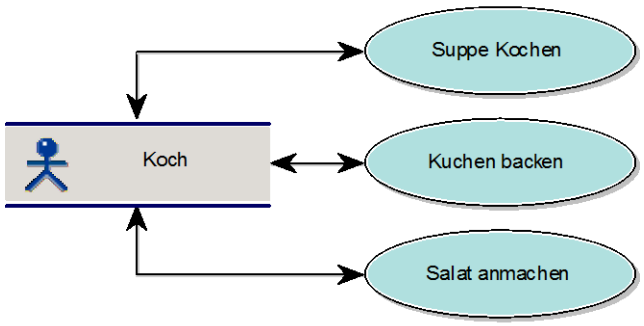

Abbildung 22: Anwendungsfalldiagramm - Koch [ObjectiF]

5.3.1.3 Das Aktivitätsdiagramm

Das Aktivitätsdiagramm ist ebenfalls ein Verhaltensdiagramm und beschreibt den Ablauf eines gewissen Vorgangs. Er kann dabei einen Anwendungsfall selbst oder einen Teil davon näher charakterisieren. Die Dinge in dem Diagramm sind eine Reihe von Aktivitäten, die sequentiell von einem Startknoten aus bis zu einem Endknoten durchlaufen werden. Eine Aktivität wird im Diagramm durch ein Rechteck mit abgerundeten Ecken dargestellt, in dem die Bezeichnung der Aktivität eingetragen wird. Der Startknoten ist ein einfacher Punkt und der Endknoten ein Punkt mit einem Kreis darum.

Die Beziehungen zwischen den Aktivitäten stellen Transitionen dar, die den Übergang von einer Aktivität zur nächsten beschreiben. Sie bilden damit einen Fluss von Informationen, der in zwei Arten untergliedert ist. Zum einen gibt es den Kontrollfluss, auf dem nur Kontroll- und Statusinformationen übertragen werden. Er wird als Linie mit einem Pfeil am Ende der Folgeaktivität gezeichnet. Der Objektfluss hingegen kann Objekte, wie z.B. Variablen oder

Klasseninstanzen transportieren. Er wird durch eine unterbrochene Linie mit Pfeil in Richtung der Folgeaktivität dargestellt, wobei sich in der Mitte der Linie ein Rechteck mit dem Namen des zu transportierenden Objekts befindet.

Es können desweiteren Verzweigungen verwendet werden, die einen alternativen Verlauf des Vorgangs mit Hilfe von Bedingungen ermöglicht. Die Verzweigungen werden durch eine unausgefüllte Raute symbolisiert, wobei eine Transition in die Verzweigung eingeht und den Weg in zwei ausgehende Transitionen aufteilt. Die Bedingungen werden in eckigen Klammern neben dem jeweiligen alternativen Wegstrang geschrieben. Transitionen können auch über einen Synchronisationsbalken vereinigt bzw. aufgeteilt werden. Die Synchronisation kann dabei noch zwischen *Und*, *Oder* bzw. *Exklusiv-Oder* unterschieden werden. Abbildung 23 zeigt ein Aktivitätsdiagramm eines Arbeitsvorgangs des schon bekannten Kochs.

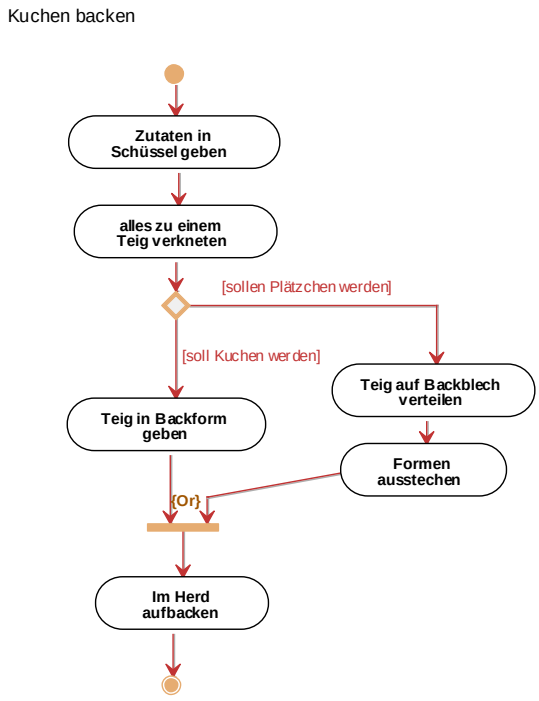

Abbildung 23: Aktivitätsdiagramm - Kuchen backen [ObjectiF]

5.3.1.4 Das Zustandsdiagramm

Das Zustandsdiagramm ähnelt sehr dem Aktivitätsdiagramm, da es auch den Ablauf eines Vorgangs beschreibt. Allerdings sind die Dinge dabei keine Aktivitäten sondern Zustände. Es ist auch so, dass das Aktivitätsdiagramm als eine Sonderform des Zustandsdiagramms bezeichnet wird. Im Gegensatz zum Aktivitätsdiagramm wird beim Zustandsdiagram der Übergang von einem zum nächsten Zustand nicht nach Ablauf aller seiner Aktionen vollzogen. Vielmehr ist ein auslösendes Ereignis notwendig, um den Zustandswechsel anzustoßen.

Die Zustände werden hier ebenfalls durch Rechtecke mit abgerundeten Ecken dargestellt, in dem der Name des Zustands eingetragen wird. Es gibt gleichfalls Start- und Endknoten, die auch wie im Aktivitätsdiagramm gezeichnet werden. Beziehungen sind wieder Transitionen, allerdings wird nicht zwischen Kontroll- und Objektfluss unterschieden. Die Zustände werden mit durchgezogener Linie und Pfeil in Richtung Folgezustand verbunden. Es existieren auch wieder Kontrollstrukturen wie die bedingte Verzweigung und der Synchronisationsbalken, die auch das gleiche Erscheinungsbild wie im Aktivitätsdiagramm haben. In Abbildung 24 sieht man einen Zustandsgraph aus dem Küchenbereich.

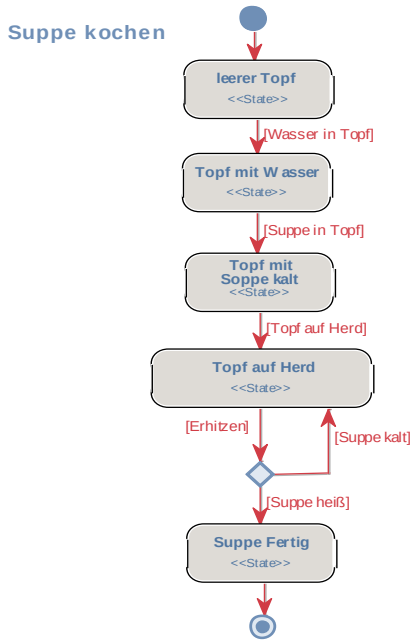

Abbildung 24: Zustandsdiagramm - Suppe kochen [ObjectiF]

6 Entwurf des Programmmodells

Nachdem in den vorangegangenen Kapiteln die theoretischen Grundlagen gelegt wurden, soll es in diesem Kapitel um die schrittweise Schaffung eines Modells für das Abfrageprogramm gehen. Dabei werden die beschriebenen Diagramme der UML zu Visualisierung verwendet.

Für den Entwurf ist es wichtig, eine Strategie zu verfolgen, damit keine Aspekte der zu entwerfenden Programmfunktionalitäten vergessen werden. Es ist eine Datenstruktur zu entwerfen, die die vom Benutzer eingegebenen Werte speichern und anderen Funktionen zur Weiterverarbeitung zur Verfügung stellen kann. Dieser Entwurf kann durch ein Klassendiagramm visualisiert werden. Desweiteren sind alle vom Benutzer benötigten Programmfunktionen zu erfassen und in einem Anwendungsfalldiagramm darzustellen. Die einzelnen Funktionen sind zu spezifizieren und in Aktivitäts- bzw. Zustandsdiagrammen zu verfeinern. Wenn das realisiert wurde, sollte das entworfene Modell eine gute Grandlage für die Implementierung in der Programmierumgebung bilden.

6.1 Die Datenstruktur

Als erstes ist es wichtig, sich Gedanken darüber machen, welche Daten in welchem Umfang gespeichert werden müssen. Mit anderen Worten gesagt: Welche Daten werden vom Benutzer eingegeben und nach erfolgreichem Beenden des Programms in die erforderliche Skriptdatei geschrieben? In Kapitel 4 wurde ausführlich geschildert, welche Einstellungen für die Inbetriebnahme des Schwenkzyklus getätigt werden müssen. Das sind zum einen die MD, wobei für den ersten Entwurf des Programms alle MD den in der Tabelle der Inbetriebnahmeanleitung vorgeschlagenen Wert erhalten sollen. Das einzige MD, welches einen konkreten Wert aus dem Abfrageprogramm erhalten muss, ist MD 18088, das die maximale Anzahl an anzulegen SDS enthält. In einer späteren Entwicklungsstufe sollen auch noch Abfragen in das Programm eingearbeitet werden, die einige der anderen MD variieren können.

Weiterhin muss es möglich sein, vom Benutzer eine bestimmte Anzahl an SDS anzulegen und die enthaltenen Systemvariablen mit entsprechenden Werten zu belegen. Es wird also eine

Datenstruktur benötigt, die die Werte aller Systemvariablen aufnehmen kann und mit der es möglich ist, mehrere Exemplare gleichen Typs zu erzeugen, da bei einem Programmdurchlauf alle vom Bediener vorgesehenen SDS nacheinander mit Werten gefüllt werden sollen.

Als Datenstruktur für den SDS bietet sich eine Klasse an, die den Namen SDS trägt und die als Mitgliedsvariablen den kompletten Satz an Systemvariablen erhalten soll, der in einem SDS enthalten ist. Weiterhin wird ein Standardkonstruktor benötigt, der alle Mitgliedsvariablen auf einen Standardwert setzt. Diese Eigenschaft vermeidet Unstimmigkeiten, wenn zwar ein SDS angelegt wird, aber nicht mit Werten gefüllt wird. Um Zugriff auf die Mitgliedsvariablen zu bekommen ist es notwendig, für jede Variable eine Get- und eine Set-Methode anzulegen. Da es allerdings möglich sein soll, nicht nur ein SDS sondern eine beliebige Anzahl anzulegen, ist dabei ein Feld dieser Klassenobjekte vorzusehen.

Es sind noch eine Reihe von Statusinformationen für den Programmablauf zu speichern. Das sind die maximale Anzahl anzulegender SDS, eine Indexvariable, in welchem SDS gerade gearbeitet wird und ein Objekt der kompletten SDS-Datenstruktur, d.h. das Feld von Klassenobjekten des SDS, da ein Objekt der Statusinformationen von einem Eingabefenster des Abfrageprogramms zum nächsten weitergegeben werden muss.

Die Datenstruktur für die Statusinformationen könnte wieder eine Klasse sein. Der Einfachheit halber wurde für das Programm eine Struktur verwendet, die den Namen Status trägt. Mitgliedsvariablen sind die schon erwähnte Anzahl an SDS, die Indexvariable und das Feld von Objekten der Klasse SDS. Es wird hier ein Initialisierungskonstruktor verwendet, der bei Programmstart und beim Anlegen eines neuen SDS die Mitgliedsvariablen auf entsprechende Werte setzt. Zusätzlich wird ein Kopierkonstruktor benötigt, der bei der Weitergabe des Status-Objekts von einem Fenster des Programms zum nächsten alle bis dahin angelegten SDS und die Statusinformationen kopieren soll.

Aus den aufgeführten Elemente ist einem Klassendiagramm für die Datenstruktur entwickelt wurden. Abbildung 25 zeigt einen Ausschnitt daraus. Das komplette Diagramm befindet sich im digitalen Anhang unter Diagramm 1. Darauf ist auch zu erkennen, dass die beiden als Klassen dargestellten Objekte in Beziehung stehen, da das Feld von SDS-Objekten Teil des Status-Objektes ist. Hierbei lässt sich jetzt darüber streiten, ob die SDS-Objekte ohne das

Status-Objekt existieren können. Wenn man der Implementierung des Programms einmal vorgreift, werden die SDS-Objekte ohne Status-Objekt nicht benötigt und mit der Zerstörung des Status-Objektes gleichfalls eliminiert. Deshalb wurde für dieses Diagramm auch eine Komposition statt einer Aggregation als Beziehungstyp verwendet.

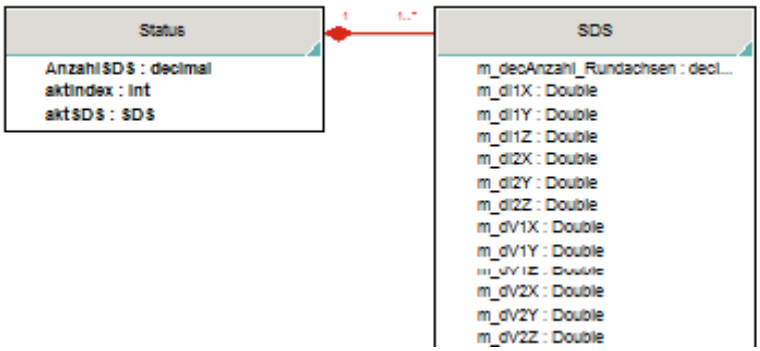

Abbildung 25: Klassendiadramm der Datenstruktur [ObjectiF]

6.2 Erfassen der benötigten Programmfunktionen

Die nächste Aufgabe auf dem Weg das Abfrageprogramm zu entwerfen, ist es, die für den Anwender des Programms wichtigen Funktionen zu spezifizieren. Damit keine Funktion vergessen wird, ist es an dieser Stelle des Entwurfs vorteilhaft, den gewünschten Ablauf des Programms festzulegen und gedanklich Schritt für Schritt durchzugehen.

Der Anwender soll in dem Programm durch eine Reihe von Seiten geführt werden, auf denen er sowohl Informationen findet, als auch die entsprechenden Werte für den SDS eingeben kann. Es muss ihm also die Möglichkeit geboten werden, von einer Seite zur nächsten vor und wenn möglich auch zurück zu gelangen. Dann sollte man von jeder Seite aus das Programm beenden können. Da es möglich sein soll, mehrere SDS anlegen zu können, sollte das Programm auch zur ersten Seite des nächsten SDS verzweigen können. Dem Anwender muss es also möglich sein, durch das Programm zu navigieren, womit schon die erste Funktion *Programmablauf bedienen* gefunden ist.

Wie schon erwähnt, soll der Anwender auch aus einigen Seiten Werte für die Variablen des SDS eingeben können. Diese Werte müssten dann über die Datenstruktur entweder im Status-

Objekt oder im entsprechenden Objekt des SDS gelesen bzw. geschrieben werden können. Die zweite Funktion nennt sich damit *SDS-Daten eingeben*.

Auf der letzten Seite des aktuellen SDS muss es möglich sein, einen neuen SDS anzulegen, wenn ein zusätzlicher SDS gebraucht wird. Dazu ist ein neues Objekt im Feld von SDS-Objekten anzulegen und dieses vorerst mit Standardwerten zu belegen. Außerdem muss kontrolliert werden, ob die maximale Anzahl an SDS überschritten wird. Für diese Aufgaben soll die Funktion *neuen SDS anlegen* zuständig sein.

Desweiteren soll man auf der letzten Seite eines jeden SDS die eingegebenen Werte noch einmal anschauen können, um sie eventuell zu korrigieren. Dazu sind alle Werte des aktuellen SDS und die Statuswerte auf einer Infoseite anzuzeigen. Diese Funktion trägt den Namen *aktuellen SDS anzeigen*.

Die wichtigste Funktion des Programms soll das Ergebnis der Abfragen in eine Skriptdatei schreiben. Es müssen dabei die fest vereinbarten MD, die maximale Anzahl an SDS und die Werte aller angelegten SDS in die Datei geschrieben werden. Der Auslöser dafür sollte sich auch im letzten Fester befinde. Die Funktion erhält den Namen *Einstellungen in Skriptdatei schreiben*.

Die eben ermittelten Funktionen des zu entwickelnden Abfrageprogramms sind in Abbildung 26 in einem Anwendungsfalldiagramm zusammengefasst und dargestellt. Man kann darauf erkennen, dass der Bediener mit allen Anwendungsfällen direkt in Beziehung steht, die Anwendungsfälle untereinander allerdings vollkommen unabhängig sind. Um wieder der Implementierung etwas vorzugreifen, ist es tatsächlich so, dass alle Funktionen voneinander unabhängig sind. Lediglich *neuen SDS anlegen* und *Programmablauf bedienen* teilen sich auf der letzten Seite den gleichen Button, beeinflussen sich jedoch nicht untereinander. Sie werden durch Abfrage des Status-Objekts unterschieden.

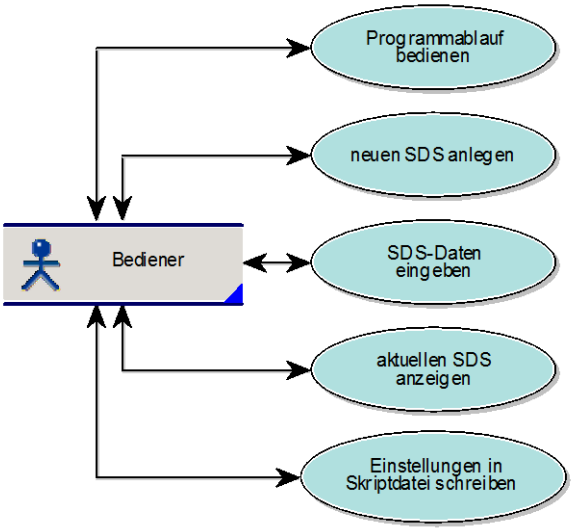

Abbildung 26: Anwendungsfalldiagramm des Abfrageprogramms [ObjectiF]

Im nächsten Schritt sind die einzelnen Anwendungsfälle näher zu betrachten, die darin enthaltenen Aktivitäten zu beschreiben und jeweils in einem Aktivitäts- bzw. Zustandsdiagramm darzustellen.

6.2.1 Programmablauf bedienen

Wie schon im vorangegangenen Abschnitt erwähnt, soll mit dieser Funktion dem Bediener die Fähigkeit gegeben werden, durch die einzelnen Seiten des Programms zu navigieren. Dazu sollten sich auf jeder Seite ein *Weiter*-, ein *Zurück*- und ein *Beenden*-Button befinden, ausgenommen der ersten, auf der kein *Zurück*-Button benötigt wird.

Ist die letzte Seite des aktuellen SDS erreicht und wurde bereits ein weiterer SDS angelegt, soll man mit *Weiter* zur ersten Seite des nächsten SDS geleitet werden. Es muss auch möglich sein, von der ersten Seite eines SDS mit der Nummer größer eins durch Betätigen von *Zurück* auf die letzte Seite des vorherigen SDS zu gelangen. Der *Beenden*-Button auf jeder Seite ermöglicht es, jederzeit das Programm abzubrechen.

Die geschilderten Vorgänge sind in einem Zustandsdiagramm dargestellt. Abbildung 27 zeigt dieses Diagramm zur besseren Übersicht mit 6 Seiten, worauf jede durch einen Zustand

repräsentiert wird und die Zustandsübergänge durch die Buttons ausgelöst werden. Es gibt hier auch einen Zustand *neuen SDS anlegen*, der nur dem Sachverhalt geschuldet ist, dass sich die beiden Funktionen auf der letzten Seite einen Button teilen. Nach dem Anlegen eines neuen SDS wird zur ersten Seite dieses neuen SDS verzweigt.

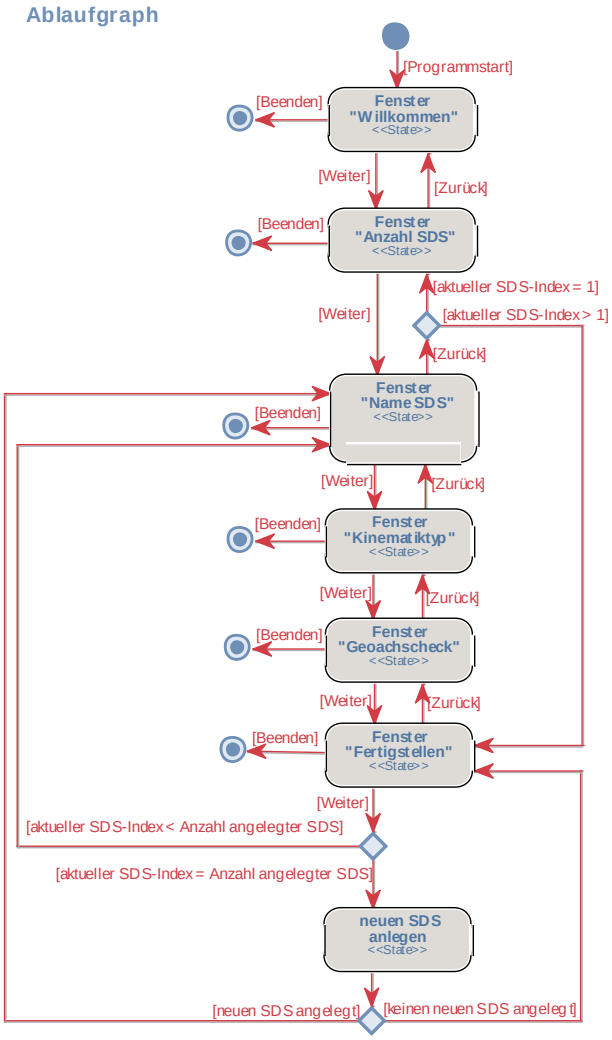

Abbildung 27: Zustandsdiagramm - Programmablauf [ObjectiF]

6.2.2 SDS-Daten eingeben

Während man durch die einzelnen Seiten des Abfrageprogramms navigiert, sollen sich auf einigen Seiten nur Informationen für die richtige Handhabung befinden, jedoch sind die meisten Seiten dazu bestimmt, die Werte für den jeweiligen SDS einzugeben. Dabei können

die Werte unterschiedlichen Typs sein, was für die Vermeidung von Fehleingaben und Wertebereichsüberschreitungen zu bedenken ist.

Bei der Eintragung in die Datenstruktur ist darauf zu achten, dass der Wert in den richtigen SDS geschrieben wird. Dazu muss der aktuelle SDS-Index aus dem Status-Objekt gelesen und das SDS-Feld damit adressiert werden. Die jeweilige Set-Methode der SDS-Klasse schreibt den Wert dann an die richtige Stelle der Datenstruktur.

In Abbildung 28 wird der Zusammenhang in einem Aktivitätsdiagramm visualisiert. Darauf ist zu erkennen, dass zwischen den einzelnen Aktivitäten ausschließlich nur Objektflüsse bestehen. Das liegt daran, dass mit dem Wert des aktuellen SDS-Index adressiert und der eingegebene Wert dann in die Datenstruktur geschrieben werden muss. Da anhand dieser Werte nicht nur Entscheidungen getroffen, sondern auch die Werte selbst der Folgeaktivität zur Verfügung gestellt werden müssen, wurden Objektflüsse zur Darstellung verwendet.

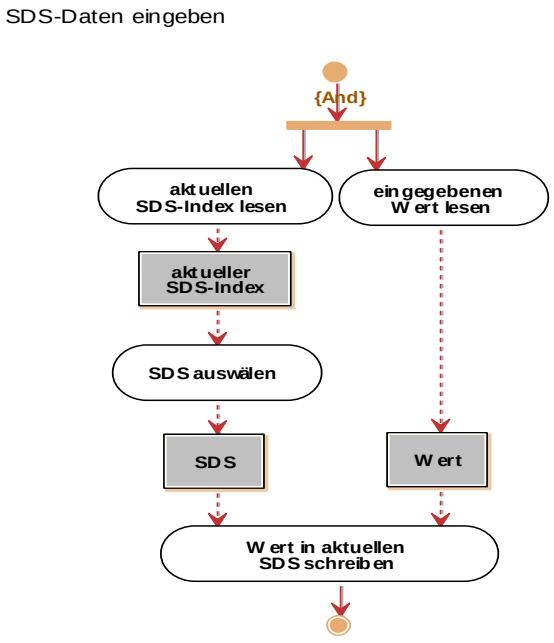

Abbildung 28: Aktivitätsdiagramm - SDS-Daten eingeben [ObjectiF]

6.2.3 Neuen SDS anlegen

Diese Funktion sollte auf der letzten Seite eines jeden SDS ausgelöst werden können, wenn man sich aktuell im höchsten bisher angelegt SDS befindet. Falls zurücknavigiert wird und schon SDS mit der Nummer größer als der aktuelle SDS-Index existieren, darf es auf der letzten Seite an der Stelle lediglich einen *Weiter*-Button geben, mit dem auf die erste Seite des nächsten SDS verzweigt wird.

Mit dieser Funktion muss wieder direkt Einfluss auf die Datenstruktur genommen werden. Dabei soll im SDS-Feld ein neues Objekt erzeugt und die einzelnen Variablen vorerst mit Standardwerten vorinitialisiert werden. Danach führt der der Programmablauf den Bediener auf die erste Seite dieses SDS.

In Abbildung 29 befindet sich das entsprechende Aktivitätsdiagramm. Darauf ist zu sehen, dass der aktuelle SDS-Index manipuliert wird, um einerseits die richtige Stelle im SDS-Feld zu adressieren und dort den neuen SDS zu erzeugen und anderseits nach dem Anlegen auch auf die richtige Seite wechseln zu können. Es wurde auch noch eine Überprüfungsfunktion eingearbeitet, die kontrolliert, ob die Nummer des neu anzulegenden SDS die maximale Anzahl an SDS übersteigt. Der Bediener soll dann entscheiden, ob er den neuen SDS anlegen will und damit die maximale Anzahl mit erhöht oder den Vorgang abbricht und auf der letzten Seite des aktuellen SDS verbleibt.

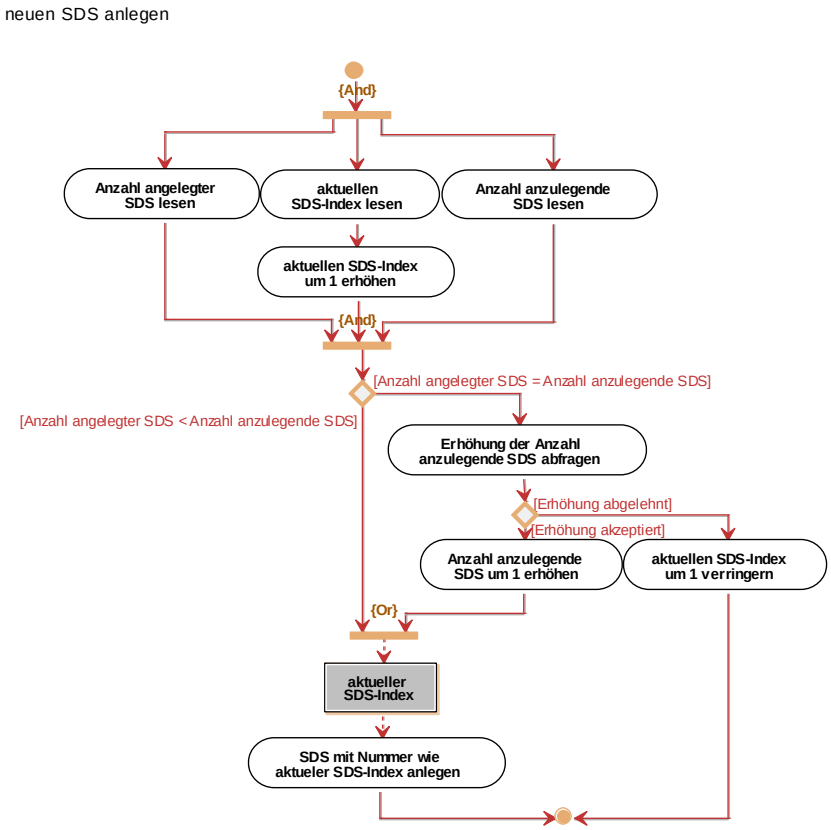

Abbildung 29: Aktivitätsdiagramm - neuen SDS anlegen [ObjectiF]

6.2.4 Aktuellen SDS anzeigen

Diese Funktion ist wichtig, um alle Werte des aktuellen SDS nach der vollständigen Eingabe nochmals kontrollieren und sie gegebenenfalls korrigieren zu können. Deshalb muss sich der auslösende Button auch hier auf der letzten Seite des jeweiligen SDS befinden. Hier ist wiederum erforderlich, dass der aktuelle SDS-Index aus dem Status-Objekt gelesen und damit der aktuelle SDS im Feld adressiert wird. Die gesamten Werte des aktuellen SDS und der Statusvariablen sollen dann auf einer Seite dargestellt werden.

In Abbildung 30 ist das dazugehörige Aktivitätsdiagramm zu sehen, auf dem wieder zu erkennen ist, dass ausschließlich Objektflüsse zwischen den Aktivitäten bestehen, um die Funktion zu realisieren.

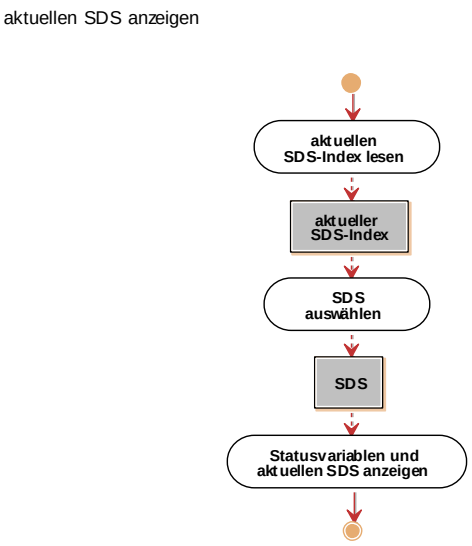

Abbildung 30: Aktivitätsdiagramm - aktuellen SDS anzeigen [ObjectiF]

6.2.5 Einstellungen in Skriptdatei schreiben

Wenn alle erforderlichen Eingaben im Programm gemacht wurden, soll diese Funktion durch Betätigen des *Fertigstellen*-Buttons das gewünschte Ergebnis liefern. Es ist eine Textdatei im ASCII-Format zu erzeugen und alle Einstellungen in der dem Update Agent verständlichen Skriptform einzutragen. Wie schon erwähnt, erhalten die MD vorerst die in der Dokumentation vorgeschlagenen Werte und sollen fest in die Skriptdatei geschrieben werden. Eine Schleife durchläuft das SDS-Feld und legt alle Werte in entsprechender Form in der Datei ab. Wenn das geschehen ist, wird die Datei geschlossen und das Programm beendet.

Abbildung 31 zeigt das Aktivitätsdiagramm für diesen Vorgang. Darauf ist zu sehen, dass noch eine zusätzliche Funktion eingebaut ist. Falls nicht aktuell auf der letzten Seite des zuletzt angelegten SDS gearbeitet wird, kann nach Klicken auf *Fertigstellen* gewählt werden, ob die SDS bis zum aktuellen oder alle bisher angelegten SDS in die Skriptdatei geschrieben werden sollen. Die Adressierung erfolgt wiederum über den Objektfluss.

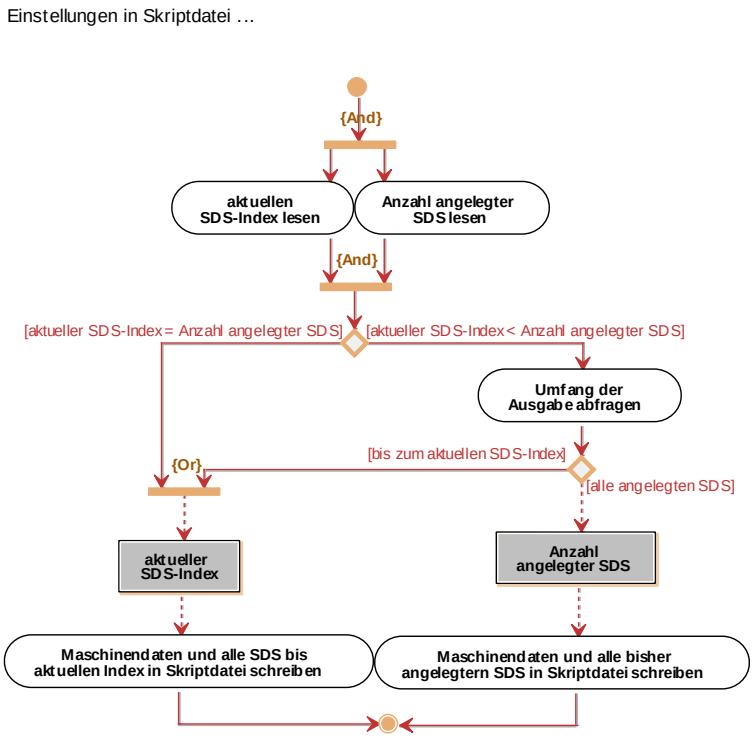

Abbildung 31: Aktivitätsdiagramm - Einstellungen in Skriptdatei schreiben [ObjektiF]

6.3 Anmerkung

Der Entwurfsprozess dieser Arbeit ist mit Sicherheit nicht vollständig und mag vielleicht etwas simpel im Gegensatz zu komplexeren Softwareprojekten wirken. Es sollte hier auch nicht jeder mögliche Fakt des Aufgabengebiets in diesem Modell umgesetzt werden. Vielmehr wurden die wichtigsten Funktionen des zu entwickelnden Programms beschrieben und in entsprechenden Diagrammen visualisiert. Zu einem späteren Zeitpunkt ist auch ein Ausbau dieser Softwarelösung vorgesehen. Das hier entwickelte Modell soll jetzt die Grundlage für die Implementierung in der Programmierumgebung bilden.

7 Implementierung des Abfrageprogramms

Aus dem gerade entwickelten Modell soll jetzt mit Hilfe der Programmierumgebung des *Microsoft Visual Studios* das Abfrageprogramm in *Visual C#* programmiert werden. Dabei ist genauso vorzugehen, wie es während des Entwurfs festgelegt wurde. Als erstes wird die für das Programm benötigte Datenstruktur angelegt. Danach sollen die einzelnen Fenster gestaltet und diese über die im Modell vereinbarte Struktur miteinander verbunden werden. Zum Schluss sind die Fenster mit den entworfenen Funktionen zu versehen.

Dafür wurde ein neues *Visual C#* - Projekt mit dem Namen *CYCLE800_IBN_Tool* angelegt. Es befinden sich am Anfang lediglich eine leeres Fenster und die Hauptprogrammfunktion *Main()* im Projekt. Das stellt den Ausgangspunkt für die Programmierung dar.

7.1 Erstellen der Datenstruktur

Für das Anlegen der Datenstruktur sollte mit den Vorgaben des Klassendiagramms gearbeitet werden. Dazu ist zuerst im Projektordner eine neue Klasse zu erstellen. Als Name für die Code-Datei wurde hier *Daten.cs* gewählt.

7.1.1 Die SDS-Klasse

In der Datei ist der Klasse der Name *SDS* zu geben und Schritt für Schritt die Elemente der Klasse zuzuordnen.

Als erstes sind die Mitgliedsvariablen hinzuzufügen. Als Namen der einzelnen Variablen wurde eine Kurzform der Beschreibungen der Systemvariablen aus Tabelle 8 im digitalen Anhang verwendet. Den in Tabelle 8 in Spalte *Sprach-Format* vorkommenden Datentypen musste ein entsprechendes Äquivalent in C# gesucht werden. Beispielsweise wurde dem Datentyp *REAL* aus der Tabelle der Datentyp *Double* in C# zugeordnet. Für die Variable *m_ekinematik* sollte ein Aufzählungsdatentyp vereinbart werden, da sie nur die Werte ‘T‘, ‘P‘ oder ‘M‘ annehmen kann. Zusätzlich ist noch eine Variable für die Anzahl an Rundachsen zu

erstellen und mit dem Datentyp *int* zu versehen. Sie dient allerdings nur für programminterne Statusabfragen, darf aber dem Status-Objekt nicht direkt angehören, da jeder SDS unterschiedlich viele Rundachsen besitzen kann. Alle Mitgliedsvariablen sind mit dem Sichtbarkeitsmerkmal *private* zu versehen. In Abbildung 32 ist ein Ausschnitt aus der Klassendefinition zu sehen, der den Klassenkopf und einen Teil der Mitgliedsvariablen zeigt. Der komplette Quellcode der Datei *Daten.cs* befindet sich im digitalen Anhang.

```
public class SDS
{
    //Mitgliedsvariablen
    private int m_decAnzahl_Rundachsen;
    private double m_dI1X;
    private double m_dI1Y;
    private double m_dI1Z;
    private double m_dI2X;
    private double m_dI2Y;
    private double m_dI2Z;
    private double m_dV1X;
    private double m_dV1Y;
    private double m_dV1Z;
    private double m_dV2X;
    private double m_dV2Y;
    private double m_dV2Z;
    private double m_dPosition_V1;
    private double m_dPosition_V2;
    private double m_dI3X;
    private double m_dI3Y;
    private double m_dI3Z;
    private double m_dI4X;
    private double m_dI4Y;
    private double m_dI4Z;
```

Abbildung 32: Klassendeklaration von SDS [MVS]

Der Standardkonstruktor muss neu geschrieben werden, weil einige Systemvariablen mit einem bestimmten Wert zu initialisieren sind. In Abbildung 33 kann man erkennen, dass die Variable *strAchsbezug_V1* z.B. den Startwert ‘X‘ erhält. Diese Initialwerte sind wiederum in Tabelle 8 im digitalen Anhang in der Spalte *Vorbelegung* notiert. Die Anzahl an Rundachsen befindet sich nicht in dieser Tabelle. Sie ist mit ‘1‘ zu initialisieren.

```
//Standardkonstruktor
public SDS()
{
    this.m_decAnzahl_Rundachsen = 1;
    this.m_dI1X = 0;
    this.m_dI1Y = 0;
    this.m_dI1Z = 0;
    this.m_dI2X = 0;
    this.m_dI2Y = 0;
    this.m_dI2Z = 0;
    this.m_dV1X = 0;
    this.m_dV1Y = 0;
    this.m_dV1Z = 0;
    this.m_dV2X = 0;
    this.m_dV2Y = 0;
    this.m_dV2Z = 0;
    this.m_dPosition_V1 = 0;
    this.m_dPosition_V2 = 0;
    this.m_dI3X = 0;
    this.m_dI3Y = 0;
    this.m_dI3Z = 0;
    this.m_dI4X = 0;
    this.m_dI4Y = 0;
    this.m_dI4Z = 0;
    this.m_strAchsbezug_V1 = "X";
    this.m_strAchsbezug_V2 = "X";
```

Abbildung 33: Standardkonstruktor von SDS [MVS]

Der Vollständigkeit halber wurde auch ein Kopierkonstruktor geschrieben, der allerdings noch nicht zur Verwendung kam. Dabei wird ihm ein SDS-Objekt übergeben und im Rumpf die Werte aus dem übergebenen Objekt auf die Mitgliedsvariablen geschrieben. Beide Konstruktoren sind mit der Sichtbarkeit *public* zu deklarieren.

Für jede Mitgliedsvariable sind schließlich noch eine Get- und eine Set-Methode zu schreiben, um eine Schnittstelle zu den mit *private* definierten Variablen zu schaffen und somit außerhalb der Klasse auf die Variablen zugreifen zu können. Die Get-Methode liest den Wert aus dem SDS-Objekt und liefert ihn als Rückgabewert. Die Set-Methode bekommt einen Wert übergeben und schreibt ihn in das SDS-Objekt. Die Methoden erhalten ebenfalls die Sichtbarkeit *public*. In Abbildung 34 sind ein Teil der Get- und Set-Methoden zur Veranschaulichung zu sehen.

```
        this.m_dI4X_Fein = savedSDS.m_dI4X_Fein;
        this.m_dI4Y_Fein = savedSDS.m_dI4Y_Fein;
        this.m_dI4Z_Fein = savedSDS.m_dI4Z_Fein;
        this.m_dV1_Fein = savedSDS.m_dV1_Fein;
        this.m_dV2_Fein = savedSDS.m_dV2_Fein;
    }

    //Mitglieds-get-Methoden
    public int get_Anzahl_Rundachsen() {return(m_decAnzahl_Rundachsen);}
    public double get_I1X() {return(m_dI1X);}
    public double get_I1Y() {return(m_dI1Y);}
    public double get_I1Z() {return(m_dI1Z);}
    public double get_I2X() {return(m_dI2X);}
    public double get_I2Y() {return(m_dI2Y);}
```

Abbildung 34: Get- und Set-Methoden von SDS [MVS]

7.1.2 Die Status-Struktur

In Kapitel 6 wurde schon beschrieben, dass für den Programmablauf einige Statusvariablen benötigt werden. Dafür muss auch eine Datenstruktur angelegt werden. Diese Datenstruktur könnte wiederum eine Klasse sein, wurde wegen des geringen Umfangs allerdings mit einer Struktur implementiert.

Diese Struktur trägt der Namen *Status* und enthält ebenfalls Mitgliedsvariablen. Aus dem Entwurf ist bekannt, dass Variablen für die maximale Anzahl an SDS und den aktuellen SDS-Index benötigt werden. Der Einfachheit halber sollen diese Variablen als *public* deklariert werden, um sich das Implementieren der Get- und der Set-Methode zu sparen.

Im Klassendiagramm war zu sehen, dass eine Kompositionsbeziehung zwischen der Status-Struktur und der SDS-Klasse besteht, da ein Status-Objekt von einem Fenster zum nächsten weitergegeben werden soll. Dabei werden die Informationen der SDS-Objekte in jedem Fenster verfügbar gemacht. Desweiteren konnte man ein Kardinalitätsverhältnis von “1:n“ zwischen Status und SDS erkennen, da vom Benutzer das Anlegen mehrerer SDS möglich ist und diese mit dem Status-Objekt mit übergeben werden sollen. Deshalb muss eine Variable in der Status-Struktur deklariert werden, die ein Feld von SDS-Objekten als Datentyp besitzt. In Abbildung 35 ist ein Teil der Strukturdeklaration zu sehen, die den Sachverhalt verdeutlicht.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace CYCLE800_IBN_Tool
{
    public struct Status
    {
        public int AnzahlSDS;
        public int aktIndex;
        public SDS[] aktSDS;
```

Abbildung 35: Status-Struktur-Declaration [MVS]

Da es bei Strukturen keinen parameterlosen Konstruktor gibt, wurde hier ein Initialisierungskonstruktor mit Übergabeparametern geschrieben. Er ermöglicht es, bei Programmstart alle Statusvariablen auf ‘1‘ zu setzten und genau einen SDS für den ersten Durchlauf anzulegen. Weiterhin sorgt er dafür, dass beim Anlegen eines neuen SDS im letzten Fenster ein neuer neben den schon vorhandenen SDS erzeugt wird, ohne diese zu beeinflussen. Er stellt auch sicher, dass man zwischen dem letzten Fenster des aktuellen und dem ersten Fenster des nächsten SDS vor und zurücknavigieren kann, ohne dass SDS-Daten verloren gehen. Abbildung 36 zeigt die Implementierung dieses Konstruktors.

```
//Initialisierungskonstruktor
public Status(int AnzahlSDS, int aktIndex, SDS[] aktSDS, int SDSFeldlaenge)
{
    this.AnzahlSDS = AnzahlSDS;
    this.aktIndex = aktIndex;
    if (aktSDS != null)
    {
        if (SDSFeldlaenge <= aktIndex)
        {
            this.aktSDS = new SDS[aktIndex];
            for (int i = 0; i < aktIndex; i++)
                this.aktSDS[i] = new SDS();
        }
        else
        {
            this.aktSDS = new SDS[SDSFeldlaenge];
            for (int i = 0; i < SDSFeldlaenge; i++)
                this.aktSDS[i] = new SDS();

        }
        for (int i = 0; i < SDSFeldlaenge; i++)
            this.aktSDS[i] = aktSDS[i];
    }
    else
    {
        this.aktSDS = new SDS[aktIndex];
        for (int i = 0; i < aktIndex; i++)
            this.aktSDS[i] = new SDS();
    }
}
```

Abbildung 36: Initialisierungskonstruktor von Status [MVS]

Der Kopierkonstruktor der Status-Struktur trägt entscheidend zur Übergabe der Informationen von einem Fenster zum nächsten bei. Dabei werden die Statusvariablen und alle bisher

angelegten SDS vom Status-Objekt des vorangegangenen Fensters auf das des aktuellen Fensters kopiert. Der Quellcode des Konstruktors ist in Abbildung 37 zu sehen.

```
//Kopierkonstruktor
public Status(Status savedStatus)
{
    this.AnzahlSDS = savedStatus.AnzahlSDS;
    this.aktIndex = savedStatus.aktIndex;
    this.aktSDS = new SDS[savedStatus.aktSDS.Length];
    for (int i = 0; i < savedStatus.aktSDS.Length; i++)
        this.aktSDS[i] = savedStatus.aktSDS[i];
}
```

Abbildung 37: Kopierkonstruktor von Status [MVS]

Die Datenstruktur ist somit komplett und es kann zur Entwicklung der eigentlichen Programmfunktionen übergegangen werden.

7.2 Entwickeln des Abfrageprogramm

Der erste Anwendungsfall im Modell beschreibt die Funktion der Programmnavigation zwischen den einzelnen Fenstern. Um das zu realisieren, sind in einem ersten Schritt die besagten Fenster zu entwerfen und deren Oberfläche zu gestalten. Es sind Felder für Texte und Überschriften aufzubringen und diese mit den entsprechenden Beschreibungen zu füllen, Hilfebilder einzufügen und anzuordnen, Eingabefelder für die SDS-Werte zu schaffen und die entsprechenden Buttons für die Navigation und die erforderlichen Funktionen anzubringen. Um das zu verdeutlichen, soll dieser Vorgang an einem Fenster beschrieben werden und als Vorlage für alle anderen Fenster dienen.

7.2.1 Gestaltung der Fenster

Das Fenster, welches beim Anlegen des Projekts erstellt wurde, soll als Start- bzw. Willkommensfenster dienen, da es bei Programmstart automatisch geöffnet wird. In ihm sollen die Überschrift, ein Textfeld, auf dem eine kurze Beschreibung des Programms zu lesen ist, ein Bild, ein *Weiter*- und ein *Beenden*-Button sowie ein Textfeld, worauf die Funktionen der Buttons erklärt werden, zu finden sein.

Um jedoch eine Schablone für die Mehrheit der Fenster zu haben, soll eines mit Eingabefeldern näher erläutert werden. Dieses Fenster trägt den Namen *Kinematiktyp* und ist von der Reihenfolge des Zustandsdiagramms her das vierte vom Startpunkt aus. Um die Übersichtlichkeit zu wahren, sollen alle Fenster über den gleichen Aufbau verfügen. Das heißt, die Elemente der Fenster sollen sich näherungsweise an der gleichen Stelle befinden.

Für den Entwurf des Fensters *Kinematiktyp* wird zuerst dem Projektordner ein neues Fenster (Windows-Form) hinzugefügt. Danach findet man wieder ein leeres Fenster vor, das zuerst auf eine bestimmte Größe gebracht werden sollte. Diese Größe ist bei allen Fenstern beizubehalten. Als nächstes sind die einzelnen Elemente in der Toolbox auszuwählen und auf das Fenster zu ziehen. Für dieses Fenster werden ein Label für die Überschrift, drei Picture-Boxen für Bilder, eine große Text-Box für den Beschreibungstext, eine kleine für die Erläuterung der Buttons, drei Buttons für *Weiter*, *Zurück* und *Beenden*, ein NumericUpDown-Feld für die Eingabe der Rundachsenanzahl, eine Combo-Box für die Auswahl des Kinematiktyps sowie zwei Label für die Beschriftung der Eingabefelder benötigt. Die Elemente sind wie in Abbildung 38 anzuordnen und zu beschriften.

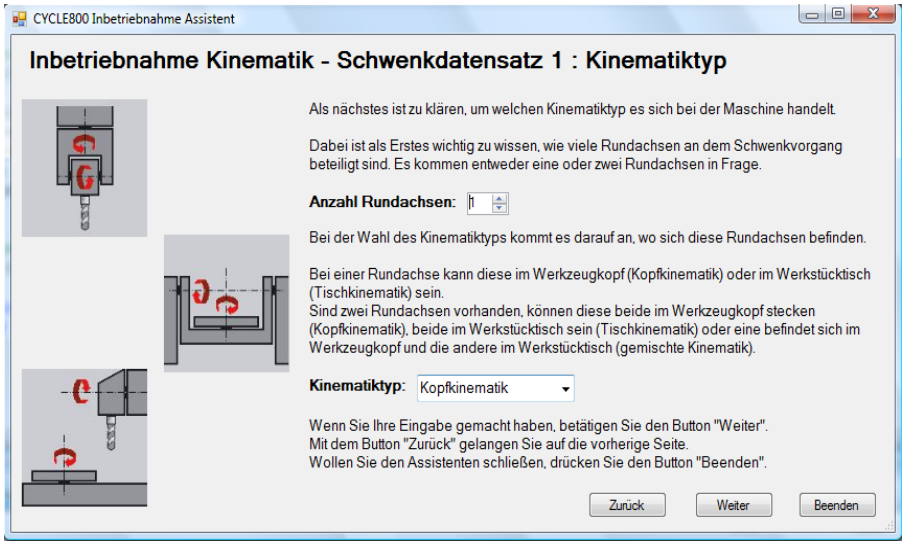

Abbildung 38: Fenster - Kinematiktyp [MVS]

Um später auf die Eigenschaften dieser Elemente besser zugreifen zu können, sind ihnen aussagekräftige Namen zu geben. Beispielsweise trägt die Combo-Box den Namen *cBKinTyp*.

Die Beschriftung erfolgt über die Eigenschaft *Text* des jeweiligen Elements. Für die bessere Lesbarkeit sind die Label fett gedruckt zu schreiben. Das lässt sich über die Eigenschaft *Font* realisieren.

Um Fehleingaben zu vermeiden, ist bei dem NumericUpDown-Feld der Minimalwert auf ‘1‘ und der Maximalwert auf ‘2‘ zu setzten, da sich auch die Anzahl an Rundachsen nur in diesem Bereich bewegen kann. Die Eigenschaften dafür sind *Minimum* und *Maximum*.

Die Combo-Box ist mit den Elementen zu füllen, die später zur Auswahl angezeigt werden sollen. Über die Eigenschaft *Items* kann man die Elemente in einer Liste eintragen. Für diese Combo-Box sind das die Elemente *Tischkinematik* und *Kopfkinematik*.

Als letztes müssen die Picture-Boxen noch ihre Bilder erhalten. Dazu gibt es die Eigenschaft *Image*, über die man Bilder in verschieden Formaten als Ressource in das Projekt importieren und der Picture-Box zuweisen kann. Wenn jede der drei Boxen ein Bild erhalten hat, ist die Gestaltungsarbeit an diesem Fenster beendet.

Dieses Vorgehen ist bei jedem Fenstern durchzuführen und Schritt für Schritt mit den entsprechenden Elementen und Informationen zu füllen. Anzahl und Aufteilung der Fenster hängen von den einzugebenden Werten der SDS sowie den für das bessere Verständnis benötigten Informationen ab und wurden im Zustandsdiagramm festgelegt.

Das erste und das letzte Fenster bilden eine Besonderheit. Auf dem ersten Fenster gibt es keinen *Zurück*-Button, da man von dem Fenster aus nicht weiter zurücknavigieren darf. Die Verbindung zum letzten Fenster des vorherigen SDS soll erst im dritten Fenster erfolgen. Das letzte Fenster besitzt zwei zusätzlich Buttons, die zwei der erforderlichen Funktionen auslösen sollen. Das sind die Buttons *SDS anzeigen* und *Fertigstellen*. Sonst haben die beiden Fenster den gleichen Aufbau wie die anderen.

7.2.2 Implementieren der Programmfunktionen

Die gerade optisch gestalteten Fenster sind als nächstes in einer bestimmten Reihenfolge aneinanderzureihen und die Möglichkeit zu schaffen, von einem Fenster zum nächsten zu

gelangen. Die Reihenfolge ist wiederum im Zustandsdiagramm genau festgelegt. Danach sind die restlichen im Modell entworfenen Funktionen des Programms zu implementieren.

7.2.2.1 Programmablauf bedienen

Die Navigation soll über die ButtonClicked-Methode der *Weiter*- und *Zurück*-Buttons erfolgen. In Abbildung 39 ist zu sehen, dass in der Methode für den *Weiter*-Button zuerst ein Objekt der Klasse des Folgefensters instanziert wird. Anschließend muss über das Objekt die Methode *Show()* aufgerufen werden, um das Folgefenster anzuzeigen. Zum Schluss wird über die Methode *Hide()* das aktuelle Fenster ausgeblendet.

```
private void btnWeiter_Click(object sender, EventArgs e)
{
    FormGeoachscheck frmGeocheck = new  FormGeoachscheck(status);
    frmGeocheck.Show();
    Hide();
}
```

Abbildung 39: ButtonClicked-Methode von Fenster *Kinematiktyp* [MVS]

Außerdem muss das im aktuellen Fenster bearbeitete Status-Objekt dem Konstroktor des Folgefensters übergeben werden, damit ihm alle bisher geschrieben Werte der Datenstruktur zur Verfügung stehen. Das bedeutet auch, dass der Standardkonstruktor jedes Fensters umgeschrieben und mit einer Objektvariablen der Datenstruktur als Übergabeparameter versehen werden muss. Im Rumpf des Konstruktors wird dann ein neues Objekt der Datenstruktur instanziert und über den Kopierkonstruktor der Status-Struktur mit den Werten des übergebenen Objekts versorgt. Abbildung 40 zeigt den Konstruktor eines Fensters. Darauf ist auch zu erkennen, dass über den Konstruktor die Startwerte der Fensterelemente gesetzt werden können.

```
Status status;

public FormNameSDS(Status savedStatus)
{
    status = new Status(savedStatus);
    InitializeComponent();
    lUeberschrift.Text = string.Format("Inbetriebnahme Kinematik - "+
                                       "Schwenkdatensatz {0} : Name Schwenkdatensatz",
                                       status.aktIndex);
    tBNameSDS.Text = status.aktSDS[status.aktIndex-1].get_Name_TCARR();
}
```

Abbildung 40: Konstruktor von Fenster *Name SDS* [MVS]

Mit dem *Zurück*-Button ist analog zu verfahren, wobei in dessen Methode das Vorgängerfenster aufgerufen werden muss. Der *Beenden*-Button, der auch auf jedem Fenster vertreten sein muss, soll das Programm ergebnislos abbrechen. Dazu ist in der ButtonClicked-Methode die Systemfunktion *Application.Exit()* aufzurufen, die jedes Fenster des Programms schließt und es anschließend beendet.

Eine Sonderstellung nehmen das dritte und das letzte Fenster ein. Wie schon erwähnt, soll das dritte Fenster in der Reihenfolge des Zustandsdiagramms das erste für die Eingabe der SDS-Daten relevante Fenster darstellen, da das erste das Willkommensfenster ist und auf dem zweiten die maximale Anzahl anzulegender SDS angegeben werden soll. Damit ist das dritte Fenster, das vom Benutzer den Namen des SDS erfragt, das erste Fenster eines jeden neu angelegten SDS. Wenn der Benutzer in diesem Fenster *Zurück* betätigt, muss der Ablauf je nachdem, in welchem SDS man sich befindet, auf das zweite Fenster des ersten oder das letzte Fenster des vorherigen SDS verzweigen. Dieser Aspekt wird durch die Abfrage und Manipulation des aktuellen SDS-Index erreicht und ist im Zustandsdiagramm genau zu erkennen. In Abbildung 41 ist der Quellcode der ButtonClicked-Methode für dieses Fenster dargestellt.

```
private void btnZurueck_Click(object sender, EventArgs e)
{
    if (status.aktIndex > 1)
    {
        status.aktIndex--;
        FormFertigstellen frmFertig = new FormFertigstellen(status);
        frmFertig.Show();
        Hide();
    }
    else
    {
        FormAnzahlSDS frmAnzSDS = new FormAnzahlSDS(status);
        frmAnzSDS.Show();
        Hide();
    }
}
```

Abbildung 41: Button *Zurück* von Fenster *Name SDS* [MVS]

Im letzten Fenster spielt der *Weiter*-Button eine Sonderrolle. Hierbei hängt die auszuführende Funktion wiederum davon ab, welcher SDS gerade bearbeitet wurde. Wie schon beschrieben, teilen sich die beiden Funktionen *Programmablauf bedienen* und *neuen SDS anlegen* im letzten Fenster einen gemeinsamen Button. Wenn man sich im zuletzt angelegten SDS befindet, hat der Button die Bedeutung *neuen SDS anlegen*, führt diese Funktion aus und navigiert dann zum ersten Fenster des neuen SDS. Hatte man aber noch Änderungen an einen

davorliegenden SDS getätigt und klickt dann im letzten Fenster auf *Weiter*, wird man auf das erste Fenster des nachfolgenden SDS geleitet, ohne einen neuen SDS anzulegen. Die Entscheidung, zu welchem Fenster der Programmablauf führt, wird wiederum durch Abfrage und Manipulation des aktuellen SDS-Index bewerkstelligt.

7.2.2.2 SDS-Daten eingeben

In diesem Abschnitt soll die Funktion implementiert werden, die es dem Benutzer ermöglicht, in den Fenstern Eingaben zu tätigen. Dafür dient wieder das Fenster *Kinematiktyp* als Vorlage für die anderen Fenster.

Auf dieses Fenster wurden zwei Eingabefelder angebracht. Das sind zum einen ein NumericUpDown-Feld und zum anderen eine Combobox.

Um jetzt die vom Benutzer während des Programmablaufs eingegebenen Werte in die Datenstruktur zu schreiben, ist als erstes ein Objekt der Status-Struktur notwendig. Da diese in jedem Fenster angelegt und über die Konstruktoren weitergeben wird, besteht in jedem Fenster Zugriff auf die aktuellen Werte.

Als nächstes ist zu überlegen, zu welchem Zeitpunkt die Daten in das Statusobjekt geschrieben werden sollen. Es wird demnach ein auslösendes Ereignis notwendig sein, dass das Schreiben der Daten anstößt. Das Klickereignis des *Weiter*-Buttons könnte dazu verwendet werden. Allerdings gehen dann die bereits eingegebenen Werte bei *Zurück* verloren. Eine bessere Variante ist es, ein Veränderungsereignis des jeweiligen Eingabefeldes zu nutzen.

Durch Doppelklick auf das NumericUpDown-Feld wird im Quelltext der Fenster-Klasse automatisch eine ValueChange-Methode erzeugt, die noch mit Programmcode gefüllt werden muss. In Abbildung 42 ist zu erkennen, dass zuerst über das Statusobjekt das SDS-Feld mit dem aktuellen SDS-Index adressiert wird, um den richtigen SDS auszuwählen und die entsprechende Set-Methode der Mitgliedsvariablen aufzurufen. Als Parameter bekommt die Set-Methode den Wert des Eingabefeldes übergeben. Da in dem Feld die Anzahl an Rundachsen eingegeben werden soll, wird auch die Set-Methode für diesen Wert verwendet.

Die Adressierung des SDS-Feldes muss um eins kleiner sein als der SDS-Index, da das SDS-Feld beim Index ‘0‘ beginnt.

```
private void nUDAnzRund_ValueChanged(object sender, EventArgs e)
{
    status.aktSDS[status.aktIndex - 1].set_Anzahl_Rundachsen((int)decimal.Round(nUDAnzRund.Value, 0));
    switch ((int)decimal.Round(nUDAnzRund.Value, 0))
    {
        case 1:
            if (cBKinTyp.Items.Contains("gemischte Kinematik"))
                cBKinTyp.Items.Remove("gemischte Kinematik");
            if (cBKinTyp.SelectedIndex == -1)
                cBKinTyp.SelectedIndex = 0;
            break;
        case 2:
            if (!cBKinTyp.Items.Contains("gemischte Kinematik"))
                cBKinTyp.Items.Add("gemischte Kinematik");
            break;
    }
}
```

Abbildung 42: ValueChanged-Methode von Fenster *Kinematiktyp* [MVS]

Die Switch-Anweisung darunter dient einem speziellen Fall. Da es bei einer Rundachse zwei und bei zwei Rundachsen drei verschiedene Kinematiktypen gibt, muss bei der Veränderung der Rundachsanzahl im Eingabefeld auch die Anzahl möglicher Auswahlbegriffe in der Combo-Box angepasst werden. Die Begriffe *Kopfkinematik* und *Tischkinematik* gibt es bei beiden Rundachsanzahlen, die *gemischte Kinematik* allerdings nur bei drei Rundachsen. Falls eine Rundachse gewählt wird, muss der dritte Begriff aus der ComboBox entfernt werden. Wenn das dritte ausgewählt war, wird die aktuelle Auswahl auf das erste Element gesetzt. Bei zwei Rundachsen wird der Begriff *gemischte Kinematik* als drittes Element hinzugefügt. Die Indizierung der Combo-Box-Elemente beginnt auch mit ‘0‘.

Mit Doppelklick auf die Combobox, wird eine SelectedIndexChanged-Methde erzeugt. Ihr Auslöser ist ebenfalls die Veränderung des eingestellten Wertes. Bei der Eingabe dieses Wertes sollte man vorher wissen, dass hier die Mitgliedsvariable *m_ekinematik* beschrieben werden soll. Dieser Variable wurde bei der Deklaration ein Aufzählungstyp zugewiesen, der die Werte ‘T‘, ‘P‘ und ‘M‘ enthält. Da dieser Typ auch über Index angesprochen werden kann und die Anordnung der Kinematiktypen in der Combo-Box mit der Anordnung der Buchstaben im Aufzählungstyp übereinstimmen, wird mit dem Index der Combo-Box der Wert des Aufzählungstyps ausgewählt.

In Abbildung 43 ist zu sehen, dass das SDS-Feld wieder über den aktuellen SDS-Index adressiert und die Set-Methode für den Kinematiktyp aufgerufen wird. Die Methode bekommt

den Index der Combo-Box als Parameter übergeben. Allerdings muss dabei ein Typecast in den Datentyp der Aufzählung gemacht werden.

```
private void cBKinTyp_SelectedIndexChanged(object sender, EventArgs e)
{
    status.aktSDS[status.aktIndex - 1].set_Kinematik((SDS.Kinematik)cBKinTyp.SelectedIndex);
}
```

Abbildung 43: SelectedIndexChange-Methode von Fenster *Kinematiktyp* [MVS]

Auf diese Weise sollen jetzt alle Eingabefelder der betreffenden Fenster mit den entsprechenden Funktionen versehen werden. Als kleiner Bonus werden die Eingabefelder mit den bereits eingegebenen SDS-Werten versorgt. Der Konstruktor der Fenster liest dabei die Werte aus der Datenstruktur und schreibt sie in die jeweiligen Eingabefelder. Diese Funktion ermöglicht das Kontrollieren der Werte beim Zurückblättern auf Fenster, die man schon ausgefüllt hat.

Wurde bereits eine gewisse Anzahl an SDS angelegt und zu dem Fenster zurücknavigiert, in dem man die maximale Anzahl anzulegender SDS angeben kann, gibt es noch eine Sicherheitsfunktion. Wird dort die maximale Anzahl anzulegender SDS in dem Maß verringert, dass sie kleiner ist als die schon angelegter SDS, wird der Benutzer mit einer Message-Box darauf hingewiesen. Er muss sich entscheiden, entweder die SDS mit der Nummer größer als die eingestellte Anzahl zu löschen oder den alten Wert beizubehalten. Diese Funktion soll Fehleingaben vermeiden. Es kann in der NCU nicht vorkommen, dass mehr SDS existieren als der im MD 18088 eingestellte Wert groß ist, da dieses MD auch den Speicherplatz für die SDS reserviert. Die entsprechende ValueChanged-Methode ist in Abbildung 44 dargestellt.

```
private void nUDAnzSDS_ValueChanged(object sender, EventArgs e)
{
    if (decimal.Round(nUDAnzSDS.Value, 0) < status.aktSDS.Length)
        switch (MessageBox.Show(String.Format(
            "Durch das Verkleinern der Anzahl gehen einige bereits angelegte Schwenkdatensätze (SDS) verloren.\n\n" +
            "Bei \"OK\" werden die SDS mit der Nummer größer {0} entfernt.\n" +
            "Bei \"Abbrechen\" bleibt die vorherige Anzahl an SDS erhalten.", decimal.Round(nUDAnzSDS.Value, 0)),
            "Anzahl Schwenkdatensätze",
            MessageBoxButtons.OKCancel,
            MessageBoxIcon.Warning,
            MessageBoxDefaultButton.Button1))
        {
            case DialogResult.OK:
                status = new Status(status.AnzahlSDS, status.aktIndex, status.aktSDS, (int)decimal.Round(nUDAnzSDS.Value, 0));
                break;
            case DialogResult.Cancel:
                nUDAnzSDS.Value = status.AnzahlSDS;
                break;
        }
    status.AnzahlSDS = (int)decimal.Round(nUDAnzSDS.Value,0);
}
```

Abbildung 44: ValueChanged-Methode von Fenster *Anzahl Kinematiken* [MVS]

7.2.2.3 Neuen SDS anlegen

Diese Funktion wurde beim Programmablauf schon angedeutet. Auslöser ist der *Weiter*-Button des letzten Fensters, wenn der zuletzt angelegten SDS aktuell bearbeitet wird. Der Button trägt dann auch die Beschriftung *neuer SDS*.

Die Entscheidung, ob ein neuer SDS angelegt werden soll oder nicht, wird im Initialisierungskonstruktor der Status-Struktur getroffen. Dazu muss der aktuelle SDS-Index mit der Anzahl bereits angelegter SDS verglichen. Wenn der aktuelle SDS-Index kleiner als die Anzahl bereits angelegter SDS ist, wird nicht im zuletzt angelegten SDS gearbeitet und der Initialisierungskonstruktor erfüllt die gleiche Funktion wie der Kopierkonstruktor. Damit kann das Status-Objekt korrekt an das erste Fenster des nächsten SDS übergeben werden. Diese Funktionalität ist dem Programmablauf unterzuordnen, da dabei kein neuer SDS angelegt wird, sondern die Größe des SDS-Feldes gleich bleibt.

Wenn der aktuelle SDS-Index so groß wie die Anzahl bereits angelegter SDS ist und demzufolge den zuletzt angelegten SDS erreicht hat, erzeugt der Initialisierungskonstruktor ein Feld von SDS-Objekten, was um eins größer ist als das bisherige und kopiert danach die bereits eingegeben SDS-Daten in das neue SDS-Feld. Damit ist neben den schon ausgefüllten SDS ein neuer entstanden, der im nächsten Durchlauf bearbeitet werden kann. Zur Veranschaulichung befindet sich in Abbildung 45 nochmals der Initialisierungskonstruktor.

```
public SDS[] aktSDS;

//Initialisierungskonstruktor
public Status(int AnzahlSDS, int aktIndex, SDS[] aktSDS, int SDSFeldlaenge)
{
    this.AnzahlSDS = AnzahlSDS;
    this.aktIndex = aktIndex;
    if (aktSDS != null)
    {
        if (SDSFeldlaenge <= aktIndex)
        {
            this.aktSDS = new SDS[aktIndex];
            for (int i = 0; i < aktIndex; i++)
                this.aktSDS[i] = new SDS();
        }
        else
        {
            this.aktSDS = new SDS[SDSFeldlaenge];
            for (int i = 0; i < SDSFeldlaenge; i++)
                this.aktSDS[i] = new SDS();

        }
        for (int i = 0; i < SDSFeldlaenge; i++)
            this.aktSDS[i] = aktSDS[i];
    }
    else
    {
        this.aktSDS = new SDS[aktIndex];
        for (int i = 0; i < aktIndex; i++)
            this.aktSDS[i] = new SDS();
    }
}
```

Abbildung 45: Initialisierungskonstruktor von Status [MVS]

Eine zusätzliche Sicherheitsfunktion prüft beim Anlegen, ob die maximale Anzahl anzulegender SDS, die im zweiten Fenster eingegeben werden muss, überschritten wird. Der Benutzer wird dann über eine Message-Box informiert und kann sich entscheiden, entweder die maximale Anzahl mit zu erhöhen oder den Vorgang des Anlegens abzubrechen. In Abbildung 46 ist die ButtonClicked-Methode vom *Weiter*-Button des letzten Fensters zu sehen, die diesen Zusammenhang zeigt.

```
private void btnWeiter_Click(object sender, EventArgs e)
{
    status.aktIndex++;
    if (status.aktIndex > status.AnzahlSDS)
        switch (MessageBox.Show(
            "Durch das Anlegen eines neuen Schwenkdatensatzes (SDS) wird die am Anfang eingegebene maximale Anzahl überschritten.\n\n" +
            "Bei \"OK\" wird die maximale Anzahl mit erhöht.\n" +
            "Bei \"Abbrechen\" wird kein neuer SDS angelegt.",
            "Schwenkdatensatz anlegen",
            MessageBoxButtons.OKCancel,
            MessageBoxIcon.Warning,
            MessageBoxDefaultButton.Button1))
        {
            case DialogResult.OK:
                status.AnzahlSDS++;
                status = new Status(status.AnzahlSDS, status.aktIndex, status.aktSDS, status.aktSDS.Length);
                FormNameSDS frmNameSDS = new FormNameSDS(status);
                frmNameSDS.Show();
                Hide();
                break;
            case DialogResult.Cancel:
                status.aktIndex--;
                break;
        }
    else
    {
        status = new Status(status.AnzahlSDS, status.aktIndex, status.aktSDS, status.aktSDS.Length);
        FormNameSDS frmNameSDS = new FormNameSDS(status);
        frmNameSDS.Show();
        Hide();
    }
}
```

Abbildung 46: ButtonClicked-Methode *Weiter* von Fenster *Fertigstellen* [MVS]

7.2.2.4 Aktuellen SDS anzeigen

Aktuellen SDS anzeigen dient der Kontrolle der Werte des aktuellen SDS. Sie kann ebenfalls im letzten Fenster eines jeden SDS über einen separaten Button aktiviert werden, der die Beschriftung *SDS anzeigen* trägt.

Der Programmcode der Funktion wurde aus Gründen des Umfangs in die Datenstruktur als eine Mitgliedsfunktion der Status-Struktur ausgelagert und kann im Fenster über das Status-Objekt aufgerufen werden.

Die Funktion selbst besteht aus einem einzigen Message-Box-Aufruf, wobei alle Statusvariablen und die Werte des aktuellen SDS untereinander angeordnet und in der Form *Bezeichnung: Wert* dargestellt wurden. Für das Anordnen des Textes und der Variablen in der Message-Box wurde die Funktion *string.Format()* verwendet. Um die Message-Box in der Datei der Datenstruktur verwenden zu können, musste der Namespace *System.Windows.Forms* inkludiert werden. Der Quellcode der Methode befindet sich auch im digitalen Anhang in der Datei *Daten.cs*.

7.2.2.5 Einstellungen in Skriptdatei schreiben

Die wichtigste Funktion *Einstellungen in Skriptdatei speichern* liefert das Ergebnis des Programms. Es befindet sich auch ein Button im letzten Fenster, mit dem diese Funktion ausgelöst wird. Er trägt die Beschriftung *Fertigstellen*.

Der Programmcode wurde ebenfalls in die Status-Struktur ausgelagert und als Mitgliedsmethode implementiert. Damit erfolgt der Zugriff wiederum über das Status-Objekt.

Es wird vom Benutzer zuerst die Angabe eines Ablageordners gefordert. Die Methode legt dann eine Textdatei in ASCII-Format am angegebenen Ablageort mit dem Namen *skript.ini* an und öffnet sie. Dazu ist ein Objekt der *StreamWriter*-Klasse verwendet worden, dass das Inkludieren des Namespace *System.IO* erforderte. Mit Hilfe der Mitgliedsmethode *WriteLine()* werden die Einstellungen, die in einer für den Update Agent interpretierbare Notation formuliert sein müssen, zeilenweise in die Datei geschrieben. Es wurde mit den fest vereinbarten MD begonnen. Die Ausnahme bildet die maximale Anzahl anzulegender SDS, die dem MD 18088 zugewiesen werden muss. Als nächstes wird das Feld von SDS mit einer Schleife durchlaufen und die Werte zeilenweise den entsprechenden Systemvariablen zugewiesen. Am Ende schließt die Methode die Skriptdatei. Ihr Quellcode ist wiederum in der Datei *Daten.cs* im digitalen Anhang zu finden.

In der ButtonClicked-Methode ist noch eine Zusatzfunktion eingebaut. Fall aktuell nicht im letzten Fenster des zuletzt angelegten SDS gearbeitet wird, öffnet sich wiederum eine Message-Box. Der Benutzer muss sich entscheiden, entweder alle SDS bis zum aktuellen oder alle bisher angelegten SDS in die Datei zu schreiben. Der Programmcode dieser Methode ist in Abbildung 47 dargestellt.

```
private void btnFertigstellen_Click(object sender, EventArgs e)
{
    Hide();
    if (status.aktIndex < status.aktSDS.Length)
        switch (MessageBox.Show(string.Format(
            "Sollen nur die Schwenkdatensätze (SDS) bis zur aktuellen Nummer in die Skript-Datei geschrieben werden?\n\n" +
            "Bei \"Ja\" werden die SDS bis zur Nummer {0} geschrieben.\n" +
            "Bei \"Nein\" werden alle bisher angelegten SDS geschrieben.", status.aktIndex),
            "Skript-Datei erzeugen",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question,
            MessageBoxDefaultButton.Button1))
        {
            case DialogResult.Yes:
                status.CreateIniFile(status.aktIndex);
                break;
            case DialogResult.No:
                status.CreateIniFile(status.aktSDS.Length);
                break;
        }
    else
        status.CreateIniFile(status.aktSDS.Length);
    Application.Exit();
}
```

Abbildung 47: ButtonClicked-Methode *Fertigstellen* von Fenster *Fertigstellen* [MVS]

7.3 Anmerkung

Nachdem diese Funktionen alle implementiert wurden, ist das Programm vorerst komplett. Die im Entwurf modellierten Aspekte konnten in das Programm integriert werden. Es sind auch einige zusätzliche Sicherheits- und Zusatzfunktionen eingebaut worden, um mögliche Fehleingaben zu vermeiden und den Komfort zu verbessern. Es gibt aber mit Sicherheit noch Potential, das Programm weiter zu entwickeln.

8 Programmtest

In diesem Kapitel soll es darum gehen, die Funktionalität des Programms zu testen. Dazu muss eine geeignete Strategie gefunden werden, um die entwickelten Programmfunktionen nach bestimmten Kriterien zu untersuchen, die korrekte Abarbeitung zu kontrollieren sowie mögliche Fehlerquellen aufzudecken.

Da das Programm nicht aus einzelnen Komponenten besteht, muss es im Ganzen getestet werden. Man könnte jedes Fenster als ein Modul ansehen und einzeln testen. Da die Elemente der Fenster überschaubar und die möglichen Fehleingaben durch den Benutzer damit begrenzt sind, soll auf diesen Ansatz verzichtet werden. Es ist ratsam, den Test des Programms nach den entworfenen Programmfunktionen zu unterteilen, einzeln zu untersuchen und zu bewerten.

Es wird dabei nicht auf die Gestaltung der Fenster, sondern auf die reine Funktionalität Wert gelegt. Dabei können auch nur wenige Aussagen über die Qualität der Datenstruktur getroffen werden, da dafür keinerlei Vorgaben vorhanden waren.

Wenn die Funktionen des Programms untersucht wurden, ist schließlich noch das Ergebnis zu betrachten. Die Skriptdatei muss im Update Agent syntaktisch korrekt erkannt und auch ohne Fehler abgearbeitet werden. Das erzeugte Archiv sollte sich in eine Teststeuerung einlesen lassen. Außerdem ist es ratsam, das Archiv mit einem Referenzarchiv in *UPDiff* zu vergleichen, um den dafür vorgesehenen Verwendungszweck des Programms zu prüfen.

8.1 Test der Programmfunktionen

Zu Beginn sind die Programmfunktionen zu kontrollieren. Dabei werden die Tests in die fünf entworfenen Programmfunktionen unterteilt und diese getrennt voneinander untersucht.

Es ist zuerst eine Beschreibung der getesteten Funktionen zu geben und diese sind zu bewerten. Zum Schluss sollen die Ergebnisse in einer Tabelle dargestellt werden.

8.1.1 Programmablauf bedienen

Als erstes muss das Programm kompilieren werden. Es sind keine syntaktischen Fehler vorhanden, wenn dieser Vorgang ohne Fehlermeldungen beendet wird. Das Programm sollte mit Hilfe des Debuggers gestartet werden, damit während der Laufzeit beispielsweise Zugriffsverletzungen auf die Datenstruktur erkannt werden.

Der erste Test für diese Programmfunktion betrifft die Navigation über die *Weiter*- und *Zurück*-Buttons. Es sollten in jedem Fenster jeweils einmal die beiden Buttons betätigt und untersucht werden, ob mit *Weiter* zum nächsten und mit *Zurück* zum vorherigen Fenster verzweigt wird. Außerdem darf es kein Fenster geben, das nicht geöffnet wird und auch beim Betätigen der Buttons sollten keine Laufzeitfehler angezeigt werden.

Wenn man während der Navigation keine Werte in die Felder eingibt, hat die maximale Anzahl anzulegender SDS den Wert ‘1‘. Damit muss der *Weiter*-Button im letzten Fenster mit *neuer SDS* beschriftet sein. Wird er betätigt, sollte die Meldung erscheinen, dass die maximale Anzahl anzulegender SDS überschritten ist, wenn ein neuer SDS angelegt werden soll. Diese Meldung ist einmal mit *Abbrechen* zu quittieren, um zu kontrollieren, ob im aktuellen Fenster verblieben und dabei kein neuer SDS angelegt wird. Wenn der Button weiterhin mit *neuer SDS* beschriftet ist, gibt es keinen neuen SDS. Man könnte auch den Button *SDS anzeigen* drücken, da auf der Infoseite ebenfalls die Anzahl angelegter SDS angezeigt wird. Sie muss noch auf ‘1‘ stehen.

Der Button ist erneut zu betätigen und die Meldung mit *OK* zu bestätigen. Man sollte sich jetzt im ersten Fenster des zweiten SDS befinden. Da in der Überschrift die aktuelle SDS-Nummer mit angezeigt wird, müsste dort eine ‘2‘ stehen. Mit *Zurück*, sollte wieder das letzte Fenster des ersten SDS erscheinen. Der *Weiter*-Button muss nun auch mit *Weiter* beschriftet sein. Wird er gedrückt, sollte ohne Meldung zum ersten Fenster des zweiten SDS verzweigt werden.

Als letzter Test für diese Programmfunktion, ist noch in jedem Fenster einmal *Beenden* zu betätigen. Dabei muss sich das Programm in jedem Fenster schließen.

Die Tests sind in Tabelle 1 im digitalen Anhang zusammengefasst und die Ergebnisse eingetragen.

8.1.2 SDS-Daten eingeben

Dieser Test wird sehr umfangreich sein, da alle Eingabefelder des Programms auf Falscheingaben, Wertebereichsüberschreitungen bzw. -unterschreitungen und Genauigkeitskriterien untersucht werden müssen. Zusätzlich ist der korrekte Zugriff auf die Datenstruktur zu überprüfen.

Um die möglichen Fehlerquellen einzuschränken, sind zuerst die verwendeten Eingabeelemente Schritt für Schritt durchzugehen und deren Fehlerpotential herauszuarbeiten.

8.1.2.1 Numerische Werte

Der größte Teil der Eingaben sind numerische Werte in Form von vorzeichenbehafteten und vorzeichenlosen Ganz- bzw. Gleitkommazahlen. Dazu werden numericUpDown-Felder verwendet. Bei diesem Element lassen sich über die Eigenschaften ein minimaler und ein maximaler Wert festlegen. Damit kann man zum einen Wertebereichsüberschreitungen und -unterschreitungen vermeiden und zum anderen je nach Sinn der Variable den zur Eingabe möglichen Zahlenbereich begrenzen.

Für die einzugebenden Werte der MD bzw. SDS werden für die Ganzzahlen *int* und für die Gleitkommazahlen *double* als Datentyp benötigt. Da die Variable des Eingabefeldes den Datentyp *decimal* besitzt, muss er bei der Übergabe konvertiert werden. Die Konvertierung von *int* nach *decimal* stellt kein Problem, da Wertebereich und Anzahl signifikanter Stellen bei *decimal* größer ist als bei *int* und damit weder Wert noch Genauigkeit verloren gehen können. In umgekehrter Richtung ergeben sich zwei mögliche Fehlerquellen. Zum einen muss der größere Wertebereich von *decimal* durch das Einschränken des Zahlenbereichs des Eingabeelements kompensiert werden. Zum anderen besteht das Problem des Rundens bei der Konvertierung, da dabei die Nachkommastellen abgeschnitten werden. Aus diesem Grund wird bei der Übergabe von *decimal*- an *int*-Variablen vorher der Wert mathematisch gerundet.

Der Datentyp *decimal* hat gegenüber *double* einen kleineren Wertebereich, die Genauigkeit ist aber größer. Da allerdings erst vom Eingabeelement in die Datenstruktur geschrieben und dann in umgekehrter Richtung gelesen wird, lässt sich dieser Unterscheid wiederum durch Beschränkung des Zahlenbereichs und der Anzahl an Nachkommastellen des Eingabefeldes lösen.

Die Eingabe von falschen Zeichen kann nicht vorkommen, weil sich über das Eingabefeld lediglich Ziffern, Komma und Minuszeichen eingeben lassen.

8.1.2.2 Zeichenketten

Der zweite Typ von Eingabefeldern ist für Namen und Bezeichnungen vorgesehen. Dazu wurden Text-Boxen verwendet, die den Datentyp *string* verwenden. Durch Angabe der Länge der Zeichenkette in den Eigenschaften des Feldes, wird die Anzahl an Zeichen beschränkt, um die Eingabe zu langer Namen zu verhindern.

Bei diesem Element kann es zu keiner Wertebereichsüberschreitung bzw. -unterschreitung kommen, da nur einzelne Zeichen eingegeben werden, die dann eine Bezeichnung ergeben. Allerdings besteht hier das Problem der Eingabe falscher Zeichen. Da für Namen laut Dokumentation lediglich Buchstaben, Ziffern und der Unterstrich als gültige Zeichen zulässig sind, wurde aus diesem Grund eine Methode geschrieben, die nach jeder Eingabe überprüft, ob es ein gültiges Zeichen war. Falls nicht, wird eine Fehlermeldung angezeigt und das fehlerhafte Zeichen wieder aus der Zeichenkette entfernt.

8.1.2.3 Weitere Eingaben

Alle übrigen Angaben werden über Combo-Boxen, Check-Boxen bzw. Radio-Buttons gemacht. Da das keine Eingabe vom Benutzer erfordert, sondern nur eine Auswahl getroffen werden muss, sind dabei auch keine Fehleingaben zu erwarten. Alle möglichen Fehler, die hier auftreten, sind auf den Programmierer zurückzuführen. Damit wurden alle Fehlerquellen, die seitens des Bedieners auftreten können, betrachtet.

8.1.2.4 Datenzugriffe

Der Zugriff auf die Datenstruktur wird während des Programmablaufs sowohl lesend als auch schreibend durchgeführt. Dazu muss in jedem Fenster ein Objekt der Status-Struktur vorhanden sein, das mit den Werten aus den vorangegangenen Fenstern versorgt wurde. Zur Adressierung des SDS-Objektes innerhalb des Strukturobjektes wird der aktuelle SDS-Index verwendet, der zuvor aus dem Statusobjekt gelesen wird. Er besitzt immer die Nummer des aktuellen SDS und wird beim Wechseln des SDS erhöht bzw. verringert. Bei der Adressierung des SDS-Feldes ist darauf zu achten, dass die übergebene Adresse um eins kleiner seiner muss als der aktuelle SDS-Index, da der Feld-Index bei null beginnt. Das Lesen der SDS-Variablen erfolgt schließlich über deren Get- bzw. Set-Methoden.

Fehler können hier lediglich durch Unachtsamkeit bei der Programmierung auftreten. Bei Schreibfehlern meldet sich der Compiler. Falsche Adressierung des SDS würde Laufzeitfehler nach sich ziehen, da möglicherweise auf ein nicht vorhandenes Feldelement zugegriffen werden soll. Die Wahl der falschen Mitgliedsmethode löst entweder eine Fehlermeldung wegen Typverletzung aus oder schreibt den Wert einfach in die falsche Variable.

Die Kompilierung wurde fehlerfrei durchgeführt. Der Debugger konnte während der Testphase keinen Laufzeitfehler feststellen. Die Adressierung des SDS und das Schreiben der Werte in die richtigen Mitgliedsvariablen ist durch die Funktion *SDS anzeigen* kontrolliert worden. Diese Funktion könnte jedoch auch Fehlern unterliegen. Die Wahrscheinlichkeit, dass alle Werte im jeweiligen SDS am richtigen Platz sind und gleichzeitig beide Funktionen genau gegensätzlich fehlerhaft arbeiten, ist allerdings minimal. Die Testergebnisse der Eingabefelder sind in Tabelle 2 im digitalen Anhang zu sehen.

8.1.3 Neuen SDS anlegen

Zum Test dieser Programmfunktion ist im Abschnitt *Programmablauf* schon einiges gesagt worden. Er ist im letzten Fenster über den *Weiter*-Button durchzuführen, der beim ersten SDS-Durchlauf mit *neuer SDS* beschriftet ist. Der aktuelle SDS-Index ist ‘1‘. Während des Programmstarts wurde ein SDS für den ersten Durchlauf angelegt und mit Standardwerten

belegt. Damit ist die Anzahl angelegter SDS ‘1‘. Diese Werte können auch über die Funktion *SDS anzeigen* kontrolliert werden.

Um einen neuen SDS anzulegen, ist der *Weiter*-Button zu betätigen und die Meldung mit *OK* zu bestätigen. Dabei sollte man sich anschließend im zweiten SDS befinden und die Anzahl bereits angelegter SDS muss jetzt auch ‘2‘ sein. Um das zu kontrollieren, navigiert man ins letzte Fenster dieses SDS und drückt dort *SDS anzeigen*.

Jetzt werden noch beide SDS mit Werten gefüllt, um sicherzugehen, dass man wirklich auf verschiedenen SDS arbeitet.

Die Testergebnisse sind in Tabelle 3 im digitalen Anhang dargestellt.

8.1.4 Aktuellen SDS anzeigen

Diese Funktion wurde schon häufig für die Tests der anderen Funktionen verwendet. Das muss aber nicht zwangsläufig bedeuten, dass sie korrekt arbeitet. Ausgelöst wird sie im letzten Fenster mit dem Button *SDS anzeigen*. Dabei geht ein Melde-Fenster auf, in dem alle Werte des aktuellen SDS und des Status-Objekts aufgelistet sind.

Mögliche Fehlerquelle kann hier auch das Lesen aus der Datenstruktur sein, die aber wieder über das Ausfüllen mehrerer SDS mit verschiedenen Werten und Kreuzvergleich minimiert werden kann. In Tabelle 4 im digitalen Anhang ist der Test dokumentiert.

8.1.5 Einstellungen in Skriptdatei schreiben

Die letzte dieser Funktionen soll jetzt das Ergebnis liefern. Sie wird im letzten Fenster über den Button *Fertigstellen* ausgelöst, erzeugt die Datei *skript.ini* im anzugebenden Verzeichnis, schreibt die MD sowie die Systemvariablen in die Datei und belegt sie mit den Werten der Datenstruktur.

Fehlerquellen könnten hier das Anlegen der Datei im Verzeichnis und wiederum das Lesen der Datenstruktur sein. Das Anlegen ist durch einfaches Ausprobieren zu kontrollieren. Dabei sollte sich die Datei *skript.ini* im angegebenen Pfad befinden. Man muss sie mit einem Editor öffnen und betrachten können, da sie im ASCII-Format erstellt wurde. Der Kreuzvergleich ist in der Datei durchzuführen. Dazu sind wieder mindestens zwei SDS anzulegen und mit unterschiedlichen Werten zu füllen. Die Systemvariablen für die SDS stehen am Ende der Datei jeweils untereinander und sind mit dem Namen *$TC_CARRm[n]* bezeichnet. Die SDS-Nummer steht in den Indexklammern und die Nummer der Systemvariablen laufend am Bezeichner. Das Testergebnis ist in Tabelle 5 im digitalen Anhang zu lesen.

Damit ist der Test der Programmfunktionen beendet und es kann in einem nächsten Schritt die Korrektheit der Skriptdatei überprüft werden.

8.2 Kontrolle der Skriptdatei

In diesem Test soll das Ergebnis des Abfrageprogramms auf Korrektheit überprüft werden. Dazu ist zuerst die erzeugte Skriptdatei in den Update Agent einzubinden und mit ihrer Hilfe ein Archiv zu manipulieren. Es soll dann zum einen versucht werden, das manipulierte Archiv in eine Steuerung einzulesen und zum anderen über *UPDiff* mit einem Referenzarchiv zu vergleichen. Dieser Test stellt sicher, dass das Abfrageprogramm die Daten richtig verarbeitet und diese entsprechend der Richtlinien zur Parametrierung eines SDS gesetzt hat und damit die Aufgabenstellung erfüllt wurde.

8.2.1 Test im Update Agent

Hier ist jetzt wie in Kapitel 4.2.3 vorzugehen. Um die volle Auswirkung der Veränderungen zu sehen, ist es ratsam, ein Archiv zu verwenden, bei dem zuvor noch kein Schwenkzyklus projektiert wurde. Dieses Archiv ist in *UPExpert* auf der Reiterkarte Paket zu importieren. Nachdem ein leerer Manipulationsauftrag angelegt wurde, muss die Skriptdatei im Komponentenfenster noch in den Auftrag eingebunden werden.

Einen Syntaxfehler der Datei könnte jetzt schon gesehen werden, da der Inhalt der Skriptdatei im Komponentenfenster erscheint, wenn die Datei *skript.ini* markiert wurde. Falls Zeilen mit roten Zeichen auftreten, sind entweder Bezeichner falsch geschrieben oder die Notation der Skriptsprache ist nicht eingehalten worden. In diesem Fall müsste die Funktion *Einstellungen in Skriptdatei schreiben* im Abfrageprogramm kontrolliert und überarbeitet werden.

Wenn jedoch alle Zeilen fehlerfrei angezeigt werden, kann das Paket an *UPShield* weitergegeben werden. Stellt der Überprüfungslauf keine Inkonsistenzen des Pakets fest, wird die exe-Datei für *UPShield* erzeugt. Sie ist als nächstes auf die PCU einer Teststeuerung zu kopieren und dort auszuführen.

Während der Paketabarbeitung in *UPShield* könnten noch semantische Fehler auftreten, die entweder die Abarbeitung mit einer Fehlermeldung abbrechen oder Einstellungen im Archiv falsch manipulieren. Der zweite Fakt würde allerdings erst bei der Kontrolle des Archivs bemerket werden. In beiden Fällen ist jedoch wieder eine Überarbeitung des Abfrageprogramms notwendig.

8.2.2 Überprüfung der Einstellungen in der Steuerung

Als Ergebnis der Paketabarbeitung sollte sich im Archivverzeichnis der PCU das manipulierte Archiv mit dem Namen UPDATE.ARC befinden, das als nächstes in die Steuerung einzulesen ist. Wenn während dieses Vorgangs oder danach keine Fehlermeldungen auftreten, wurden auch keine semantischen Fehler seitens der Steuerung festgestellt.

Zur Kontrolle der MD sind die einzelnen Werte im Bedienbereich *Inbetriebnahme* der HMI-Oberfläche mit den Wertvorschlägen aus Tabelle 7 im digitalen Anhang zu vergleichen. Die Systemvariablen des SDS können in den Inbetriebnahme-Masken des Schwenkzyklus überprüft werden. Außerdem kann man in der Maske *Kinematik* über *Datensatz sichern* eine Datei im Teileprogrammpfad der Steuerung mit dem Namen des SDS erzeugen. In Abbildung 48 ist zu erkennen, dass die Systemvariablen untereinander eingetragen wurden. Somit lassen sich die Werte der SDS nacheinander mit den Werten aus dem Abfrageprogramm vergleichen. Falls dabei Werte nicht übereinstimmen, ist wiederum das Abfrageprogramm zu überarbeiten.

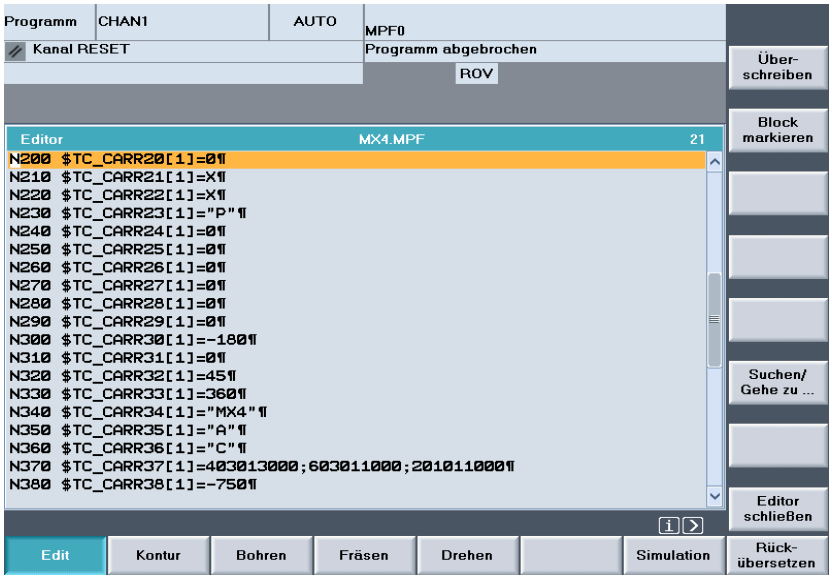

Abbildung 48: Schwenkdatensatz [Scnsht_HMI]

8.2.3 Vergleich der Archive mit UPDiff

Um in einem letzten Schritt das manipulierte Archiv mit dem Originalarchiv zu vergleichen, muss die Datei UPDATE.ARC aus dem Archivverzeichnis der Steuerung auf den Arbeitsrechner kopiert werden. Dort sollte sich auch das in UPExpert eingebundene Originalarchiv befinden. Es ist ein neuer NC-Datenvergleich zu öffnen und die beiden Archive nacheinander hinzuzufügen. Der Vergleichsfilter soll nur die Ungleichheiten anzeigen. In Abbildung 49 ist ein Teil der getätigten Veränderung zu sehen. Damit ist sichergestellt, dass das Abfrageprogramm auch korrekt gearbeitet hat. Die Testergebnisse der Kontrolle der Skriptdatei ist in Tabelle 6 im digitalen Anhang zusammengefasst.

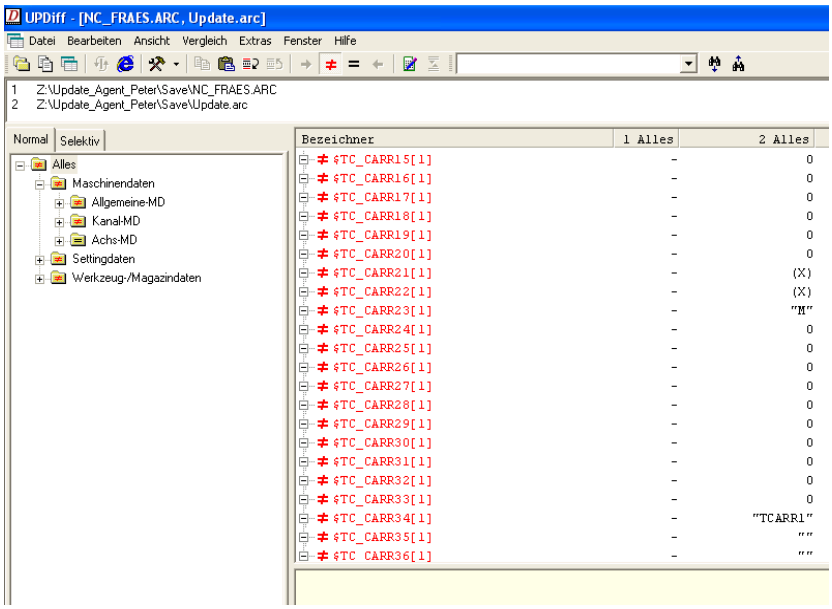

Abbildung 49: Archivvergleich nach der Manipulation [Scnsht_UA]

8.3 Anmerkung

Die durchgeführten Tests decken mit Sicherheit nicht alle möglichen Fehlerquellen ab. Es wurden aber alle vom Anwender beeinflussbaren Bedienkomponenten auf Fehleingaben getestet und die gefundenen Schwachstellen beseitigt. Außerdem ist versucht worden, mit einem kompletten Durchlauf des Kontrollvorgangs mögliche Programmierfehler auszuschließen. Damit ist der Entwicklungsprozess für diese erste Programmversion abgeschlossen und kann dem Hotline-Mitarbeiter zur Unterstützung seiner Arbeit übergeben werden. Zusätzlich ist noch eine Dokumentation anzufertigen, die die Bedienung des Abfrageprogramms erläutert.

9 Ausblick

Da für diesen ersten Entwurf lediglich ein Programmprototyp entwickelt wurde, gibt es für die Weiterentwicklung des Programms einen großen Spielraum für Verbesserung des Komforts und Erweiterung des Funktionsumfangs. Da der Schwenkzyklus selbst auch weiter entwickelt wird, sind in dieser Richtung gleichfalls Erweiterungen denkbar.

Zum einen wird bei der Inbetriebnahme des Schwenkzyklus über die Masken in der HMI beim Abspeichern zusätzlich eine Teileprogrammdatei für jeden SDS erzeugt. Diese Aufgabe könnte auch das Abfrageprogramm in Verbindung mit dem Update Agent übernehmen. Zum anderen werden derzeit alle Systemvariablen der SDS im ersten Kanal der Steuerung abgelegt. Da die SDS allerdings auch unterschiedlichen Kanälen der Steuerung zugeordnet werden können, muss durch das Abfrageprogramm auch eine Zuordnung der SDS auf Kanäle möglich sein.

Das Verbesserungspotential des Programmkomforts kann erst über Verwendung und Bewertung durch die Hotline-Mitarbeiter spezifiziert werden. Die Grenzen des Programms und Einschränkungen in der Verwendung kann nur durch die Benutzung des Programms bei Situationen des Hotline-Alltags aufgezeigt werden.

10 Zusammenfassung

Aufgabe dieser Arbeit war es, ein Abfrageprogramm zur Unterstützung der Hotline-Mitarbeiter zu entwickeln. Mit diesem Programm soll es möglich sein, eine Basis für die Überprüfung der Einstellung zur Inbetriebnahme des Schwenkzyklus CYCLE800 zu schaffen.

Es gehen laufend Anfragen bei der SINUMERIK-Hotline ein, die beinhalten, dass Kunden Schwierigkeiten haben, den Schwenkzyklus für ihre Zwecke richtig zu konfigurieren und in Betrieb zu nehmen. Die Kunden erhalten dadurch ein unerwartetes Verhalten ihrer Maschine bzw. falsche Ergebnisse. Die Schwierigkeiten der Kunden mit dem Schwenkzyklus resultieren aus der hohen Komplexität und der vielseitigen Einsetzbarkeit des CYCLE800.

Das Abfrageprogramm soll den Bediener durch eine Reihe von Fenstern führen, auf denen die nötigen Einstellungen für den Schwenkzyklus einzugeben sind. Das Ergebnis ist ein Manipulationsskript für den *SinuCom Update Agent*, der damit ein Serieninbetriebnahme-Archiv manipulieren kann. Der Kunde muss dem Hotline-Mitarbeiter das Archiv seiner Steuerung zusenden. Mit den Einstellungen aus dem Abfrageprogramm wird eine Kopie des Kundenarchivs manipuliert. Danach können beide Archive miteinander verglichen werden. Anhand der Unterschiede kann der Hotline-Mitarbeiter den Kunden besser bei der Beseitigung der Schwierigkeiten mit dem Schwenkzyklus beraten.

Zur Lösung der Aufgabe wurde zuerst das Themengebiet analysiert, die Zusammenhänge zwischen der Funktion des Schwenkzyklus und den einzustellenden Parametern erläutert und die für die Umsetzung benötigten Hilfsmittel beschrieben. Danach ist ein Modell des Programms mit der Entwurfsmethode der Unified Modeling Language entwickelt worden. Mit diesem Modell konnte dann der Entwurf in der Programmierumgebung implementiert werden. Zum Schluss wurde das Programm auf mögliche Fehlerquellen und Falscheingaben durch den Bediener untersucht und gefundene Schwachstellen beseitigt, um die korrekte Abarbeitung sicherzustellen.

Das Ergebnis dieser Arbeit ist ein funktionsfähiges Abfrageprogramm und eine Dokumentation der Bedienung, die den Hotline-Mitarbeitern zur Verfügung gestellt werden.

11 Anhang

Die Tabellen aus der Inbetriebnahmeanleitung des Schwenkzyklus CYCLE800 und die Ergebnisse aus den Tests sowie das vollständige Klassendiagramm sind im digitalen Anhang zu finden. Dazu wurde eine CD mit den genannten Dokumenten erstellt und dieser Arbeit beigefügt.

12 Literaturverzeichnis

[Doc_CYCLE800] Screenshot der Inbetriebnahmeanleitung des CYCLE800 aus dem Programmierhandbuch Zyklen

[MTS_Fräs] Screenshot einer Frässimulation, MTS - CNC - Simulatoren TOPTURN und TOPMILL (Demoversion). MTS Mathematisch Technische Software-Entwicklung GmbH, 2006

[MVS] Screenshots von Quellcode und Fenstern des Abfrageprogramm aus dem *Microsoft Visual Studio 2008* heraus

[ObjectiF] Diagramme des Programmmodells, erzeugt mit dem Software-modellierungstool ObjectiF der Firma microTOOL

[Scnsht_ARC] Screenshots eines Kunden-NC-Archivs, betrachtet mit dem Wordpad

[Scnsht_HMI] Screenshots von der HMI-Oberfläche einer Siemens SINUMERIK 840D Powerline

[Scnsht_UA] Screenshots von der Oberfläche und den Menüs des *SinuCom Upade Agent*

[Script_SE] Screenshot eines Diagramms aus dem Skript der Vorlesung Software-entwicklung von Prof. Andreas Munke

Internetquellen zum Thema UML:

UML-Tutorial von der Otto-von-Guericke-Universität Magdeburg, Online im Internet: http://www-ivs.cs.uni-magdeburg.de/~dumke/UML/inhalt.htm

Wikipedia Artikel: Unified Modeling Language, Online im Internet: http://de.wikipedia.org/wiki/Unified_Modeling_Language (Stand: 23.08.2009)

Die Diplomarbeit ist mit Hilfe dieser Dokumentationen erstellt worden:

Vorlesungsskript Softwareentwicklung von Prof. Dr.-Ing. Andreas Munke, Staatliche Studienakademie Glauchau, Studienrichtung Informationstechnik

Programmierhandbuch Ausgabe 04/2006,
SINUMERIK 840D sl/840D/840Di sl/840Di/810D Zyklen.
Bestellnummer 6FC5398-3BP10-0AA0, Siemens AG, 2006

Anleitung zum Praktikum SinuCom Update Agent, Ausgabe 2008,
Bestandteil der Dokumentation von *SinuCom Update Agent* und
erhältlich als Tutorial.de.pdf, Siemens AG 2008

Skriptbeschreibung, Ausgabe 2008,
Bestandteil der Dokumentation von *SinuCom Update Agent* und
erhältlich als Skriptbeschreibung.pdf, Siemens AG 2008

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine Diplomarbeit mit dem Thema

Erstellen eines Programmprototyps zur Erzeugung eines Manipulationsskripts für den *SinuCom Update Agent* zur Konfiguration des Schwenkzyklus CYCLE800 für Standard-Kinematiken von Kundenmaschinen

ohne fremde Hilfe angefertigt habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und

3. dass ich meine Diplomarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

_____ _____

Ort, Datum Unterschrift