XCTL-Projekt
Software-Sanierung

Projektseminar

Humboldt-Universität Berlin
Institut für Informatik
Prof. Dr. Klaus Bothe

Vortrag: David Damm

Gliederung

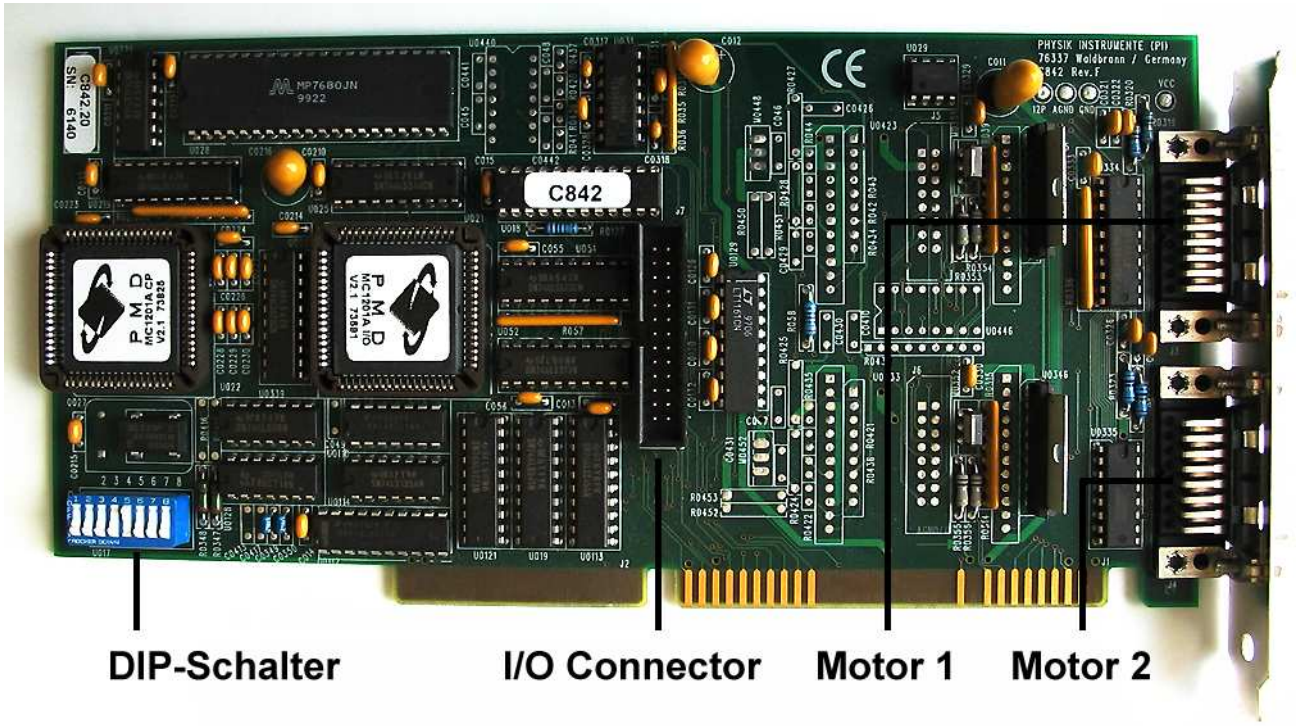
1. Studienarbeit: Einbindung des Motorcontrollers C-842 in das XCTL-System

2. Diplomarbeit: Vom Reverse-Engineering zur Programmerweiterung: Realisierung einer erweiterten Skriptsprache in einem Software-Altsystem zur Kristallanalyse

Studienarbeit

- Ansteuerung der Motoren über Motorcontroller-Karten (C-812, C-832)
- Erweiterung der XCTL-Software durch Integration des aktuellen Motorcontrollers C-842
- Ziel:
  - Erweiterung um C-842
  - gleiche Funktionalität des Gesamtsystems sicherstellen
  - möglichst geringer Aufwand

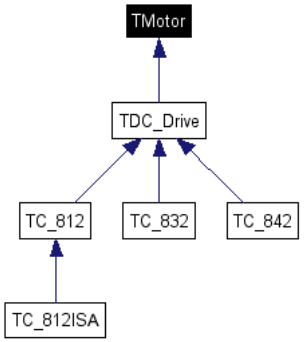
Motorcontroller



Motorcontroller-Karte C-842

Analyse

- Untersuchung des vorhandenen Quellcodes und der Dokumentationen (Klassen-struktur, Abschätzen des Aufwands, Vergleich C-832 und C-842)

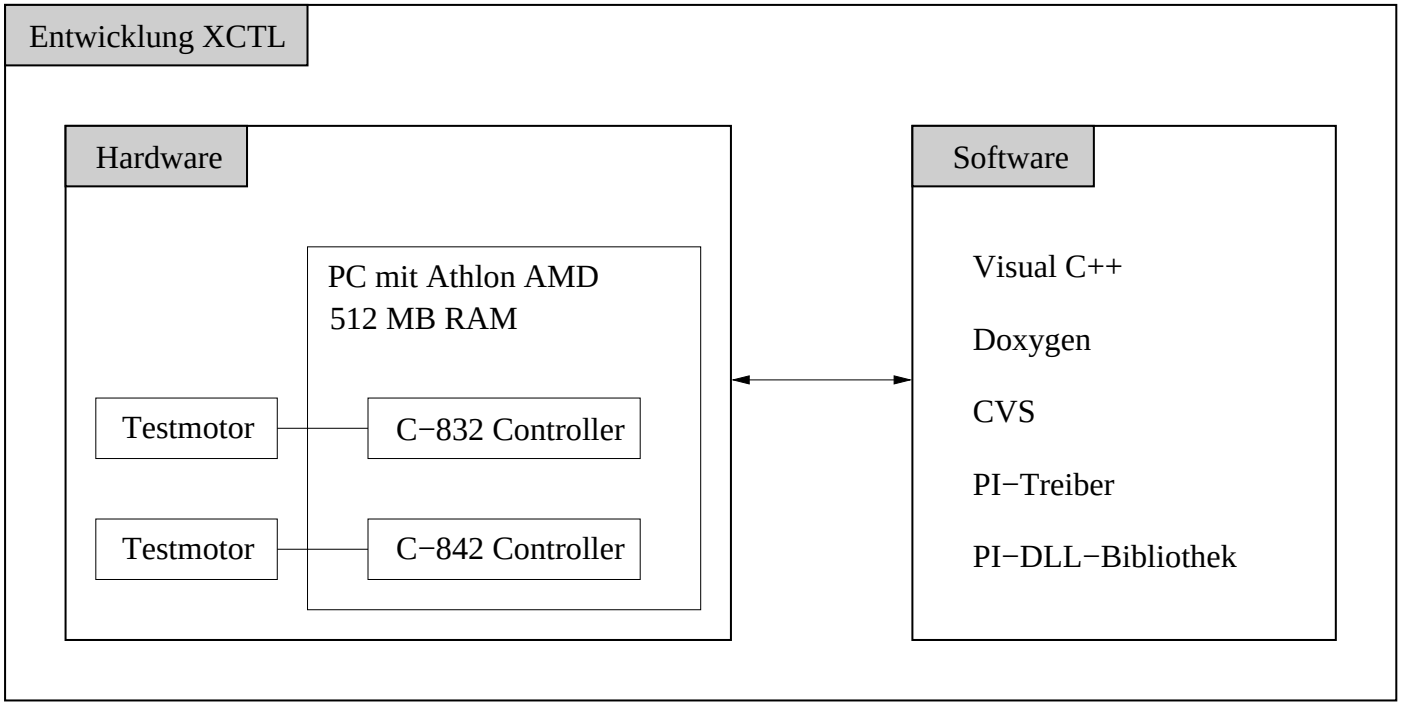


- Erstellen einer neuen Motorklasse und eines neuen Motorcontrollers

Implementation

- Kopieren beim Motor C-832
- Einbinden der DLL, die die Verbindung zwischen Software und Hardware realisiert (bereit gestellt von der Firma PhysikInstrumente)
- Neuimplementation aller Funktionen der Klasse TC_842
- Motorcontroller für C-842 wird überflüssig
- Dokumentation und Test

Entwicklung



Schematische Entwicklungsumgebung

Test

- während der Entwicklungsphase
- Regressionstest mit ATOS
  - vorhandene Testfälle für Motorenkomponente erfolgreich
  - Probleme/Fehler bei Durchführung einiger Testfälle
    - Protokollbuch: fehlende Buttons und Dialoge
    - Justage: MessageBoxen werden nicht gefunden
    - zu löschende Dateien nicht vorhanden

Erweiterung um neue Motorcontroller

- Vorgehen nach gleichem Schema wie im Abschnitt der Studienarbeit zur Einbindung des C-842 beschrieben
  - Änderungen an Klassen
  - Änderungen an Dateien
- Anwendung auf Motorcontroller C-844 denkbar

Fragen?

Fragen zur Studienarbeit...

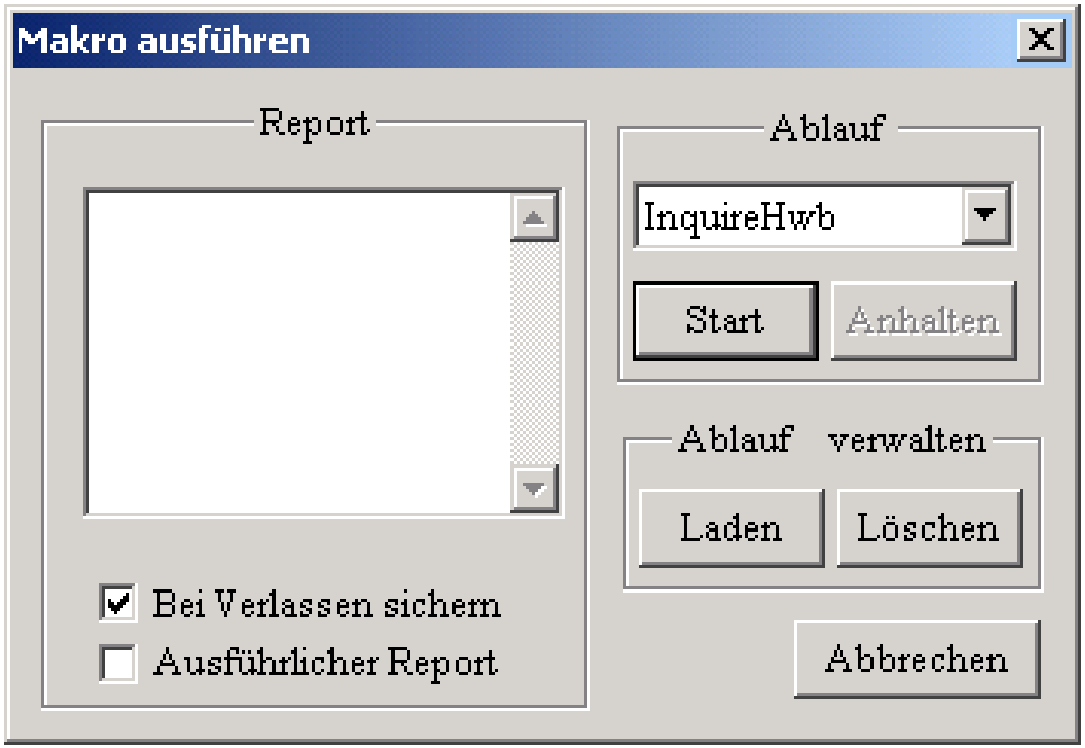
Diplomarbeit

Vom Reverse-Engineering zur Programmerweiterung: Realisierung einer erweiterten Skriptsprache in einem Software-Altsystem zur Kristallanalyse

Makrosprache

- Ablaufsteuerung für einen Versuch
  - automatische Ansteuerung der Motoren
  - Messungen durchführen
- Abarbeitung in Form eines Skripts (definiert wiederholbar, separate Datei)
- Verwendung innerhalb des Programms, aber auch durch den Benutzer direkt ausführbar

Dialogbox der alten Makrosprache



Makro ausführen

Syntax der alten Sprache (Common)

Makrodatei = {Makro "\n"}+.

Makro = CommonBlock "\n" Befehlsblock.

Commonblock = "[Common]" "\n" Namensdef "\n" Längendef.

Namensdef = "Name" Trennzeichen Makroname.

Längendef = "Length" Trennzeichen Zahl.

Makroname = "InquireHwb" | "BatchMacro" | "MiddleOfValley" | "Test"...

- es gibt 9 definierte Makros
- Makronamen sind fest im Programm verankert

Syntax der alten Sprache (Commands)

Befehlsblock = "[Commands]" "\n" Befehlsfolge "[End]".

Befehlsfolge = {Befehl {Trennzeichen Parameter}* "\n"}+.

Trennzeichen = " " | "=" | "\t" | "\r" | "\n".

Befehl = "GotoPeak" | "ChooseAxis" | "LoadPoint" | "ShowValue" | "MoveToPoint"...

- es gibt 21 definierte Kommandos
- teilweise verwirrende Namensgebungen: LoadPoint

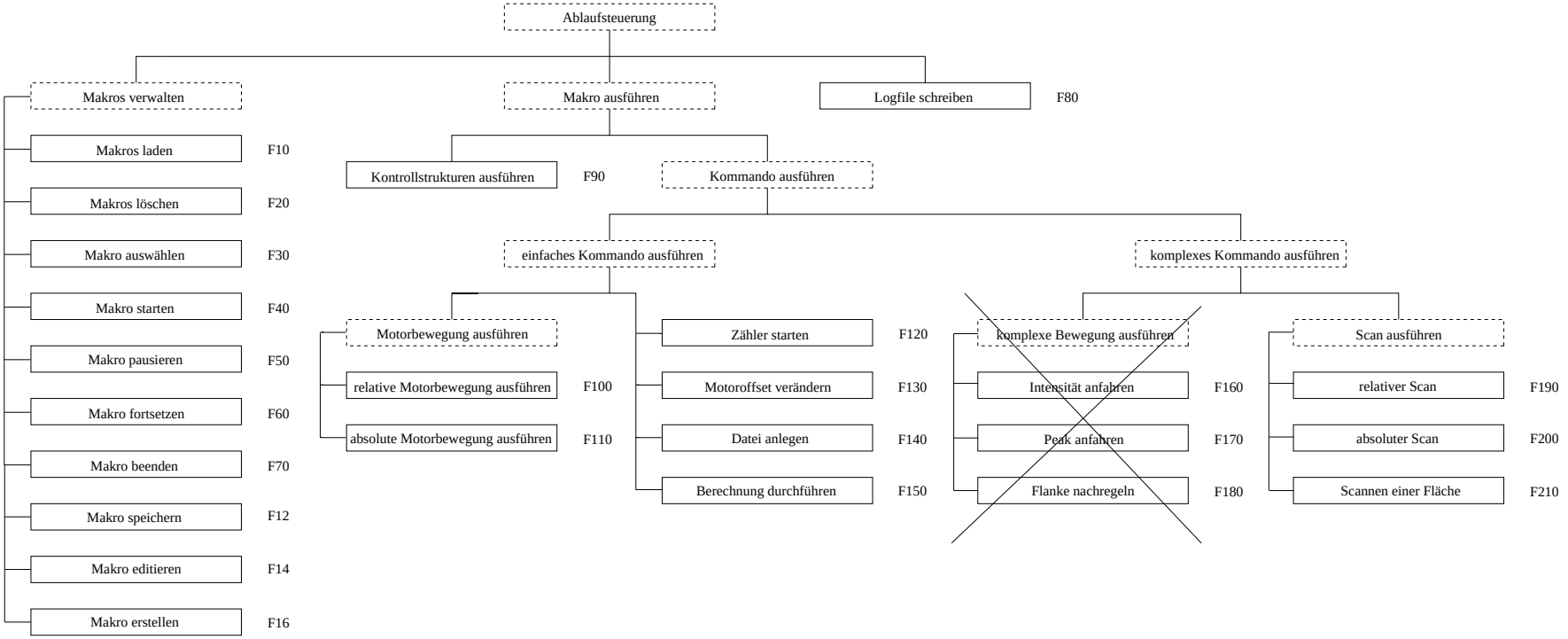
Makrodatei (Beispiel)

alte Sprache	neue Sprache (kurz)	neue Sprache (lang)
[Common]		
# Test Makro		
Name=Test		
Length=5		
[Commands]		
LoadPoint Start	get theta start	GetPosition theta start
MoveToPoint Relative 100.0	mvr theta 100.0	MoveRelative theta 100.0
ShowValue Start	print start	Print start
MoveToPoint Start	mv theta start	MoveAbsolute theta start
Stop		
[End]		

Neue Skriptsprache

- Erweiterung des Funktionsumfangs der alten Skriptsprache
- Orientierung der Namensgebung für Kommandos an der Sprache SPECTRA
  - Groß/Kleinschreibung invariant
  - Kurzform und Langform

1. Entwurf der neuen Skriptsprache



Funktionen im Pflichtenheft (1)

- Verwaltung
  - Makro laden
  - Makro löschen
  - Makro auswählen
  - Makro starten
  - Makro pausieren
  - Makro fortsetzen
  - Makro beenden
  - Makro speichern
  - Makro erstellen
  - Makro editieren

Funktionen im Pflichtenheft (2)

- einfaches Kommando
  - relative/absolute Motorbewegung ausführen
  - Zähler starten
  - Motoroffset verändern
  - Datei anlegen
  - Berechnung durchführen
- komplexes Kommando
  - relativer Scan
  - absoluter Scan
  - Scannen einer Fläche

Funktionen im Pflichtenheft (3)

- Logfile schreiben
- Kontrollstrukturen
- folgende komplexen Kommandos der alten Sprache, könnten in der neuen Skriptsprache als Makros realisiert werden:
  - Intensität anfahren
  - Peak anfahren
  - Flanke nachregeln

Kontrollstrukturen und Variablen

- Erweiterung der alten Skriptsprache um Kontrollstrukturen
  - if-else-Konstrukt
  - Möglichkeit einer Wiederholung durch while-Schleife
  - break, continue innerhalb der Schleife
- Einführung von Variablen
  - können durch den Nutzer selbst definiert werden

Kontrollstrukturen (Beispiel)

Beispiel 1	**Beispiel 2**
start = 0.0	i = 0
get theta start	while i < 20
if start > 0	mvr omega 10
mv theta 50.0	get omega x
else	print x
mv theta -50.0	i = i + 1
endif	endwhile

Laufzeitanalyse

- ob ein Makro syntaktisch korrekt ist, wird erst zur Laufzeit zeilenweise überprüft
  - Ausnahme: vor der Ausführung wird nur getestet, ob die Schachtelung der Blöcke fehlerfrei ist (if, else, endif, while, endwhile)
- Variablen werden bei ihrer ersten Verwendung deklariert
  - alle Bezeichner, die keine Schlüsselworte, Kommandos oder Antriebe sind, werden als Variable interpretiert
  - eine Variable kann nur den Typ integer oder double annehmen

Ziel

- Realisierung der neuen Skriptsprache als eigenständige Applikation unter Verwendung der Motors.dll (gekapselte C-Schnittstelle erlaubt Zugriff auf die Motoren)
- zusätzlich Integration der Skriptsprache in das XCTL-System (beide Sprachen werden parallel verwendet)
- Verbesserung der Nutzeroberfläche für die neue Skriptsprache, die neben der alten Dialogbox erstellt wird
- Umsetzen der Verwaltung, der Kontrollstrukturen und einfachen Kommandos
- Wunsch: komplexere Kommandos realisieren (je nach Aufwand)

Probleme/Fragen

- Herstellen eines Kompromisses der geforderten Funktionalität und dem Aufwand
- Vereinigung von Teilen der alten Sprache (z.B. ist GotoIntensity überflüssig) und den neuen Anforderungen (Berechnungen automatisch ausführen)
- Zugriff auf Motorfunktionen über C-Schnittstelle möglich, aber wie können die Detektoren von außerhalb des Programms für Messungen angesprochen werden?

Ende