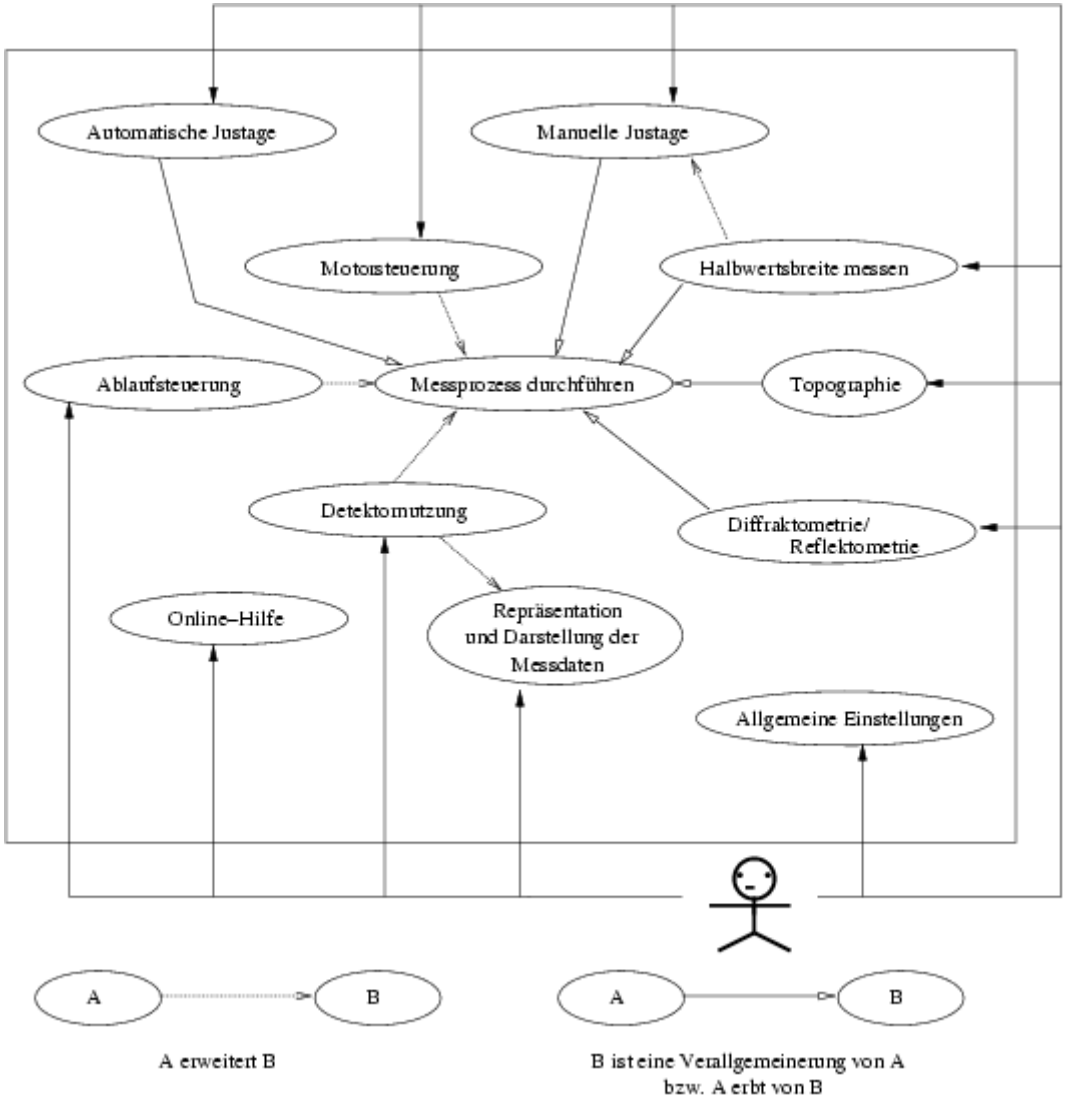
Vom Reverse Engineering zur Programmerweiterung: Realisierung einer erweiterten Skriptsprache in einem Software-Altsystem zur Kristallanalyse

David Damm

July 5, 2007

XCTL-System


Automatische Justage
Manuelle Justage
Motorsteuerung
Halbwertsbreite messen
Ablaufsteuerung
Messprozess durchführen
Topographie
Detektornutzung
Diffraktometrie/
Reflektometrie
Online–Hilfe
Repräsentation
und Darstellung der
Messdaten
Allgemeine Einstellungen
A
B
A erweitert B
A
B
B ist eine Verallgemeinerung von A
bzw. A erbt von B

Motivation und Aufgabenstellung

- funktionales und komfortables Hilfsmittel in Form einer Skriptsprache gefordert
- einfache und intuitive Bedienung
- Erleichterung bei ständig wiederkehrender Durchführung von Messprozessen (wiederholt und automatisch)

- Auseinandersetzung mit Compilerbau und Softwareengineering
- Skriptsprache erstellen, die dem gewünschten Funktionsumfang (Pflichtenheft) durch Physiker gerecht wird
- grafische Nutzeroberfläche unter Berücksichtigung ergonomischer Richtlinien überarbeiten
- neue Skriptsprache in das XCTL-System integrieren und dokumentieren

Gliederung der Diplomarbeit (1)

Gliederung der Diplomarbeit (2)

Gliederung der Diplomarbeit (3)

Beispiel-Skript (Idee)

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]		

Orientierung an SPEC

Contents -> mv

spec Help pages

NAME

mv and umv - move one motor

SYNOPSIS

```
mv motor user_value
umv motor user_value
```

DESCRIPTION

The mv macro moves *motor* to the user angle given by *user_value*.

The umv macro also moves the motor, but provides an updated display of the motor position on the screen while the motor is moving. The update interval, in seconds, is set by the global variable UPDATE.

EXAMPLE

```
mv th 0
```

SEE ALSO

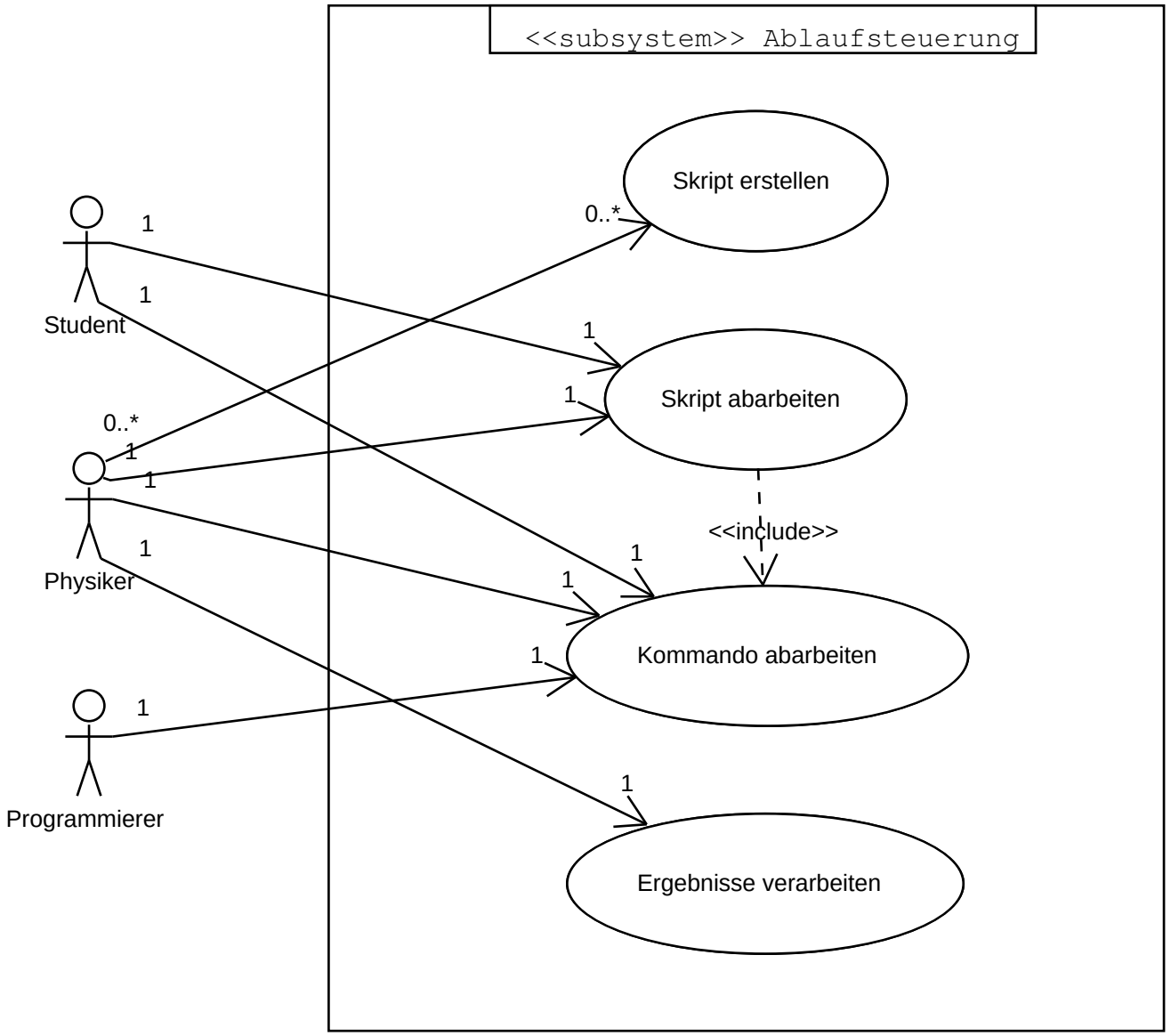
mvd mvr motors wait

... Meeting the software needs of scientists since 1985 ...

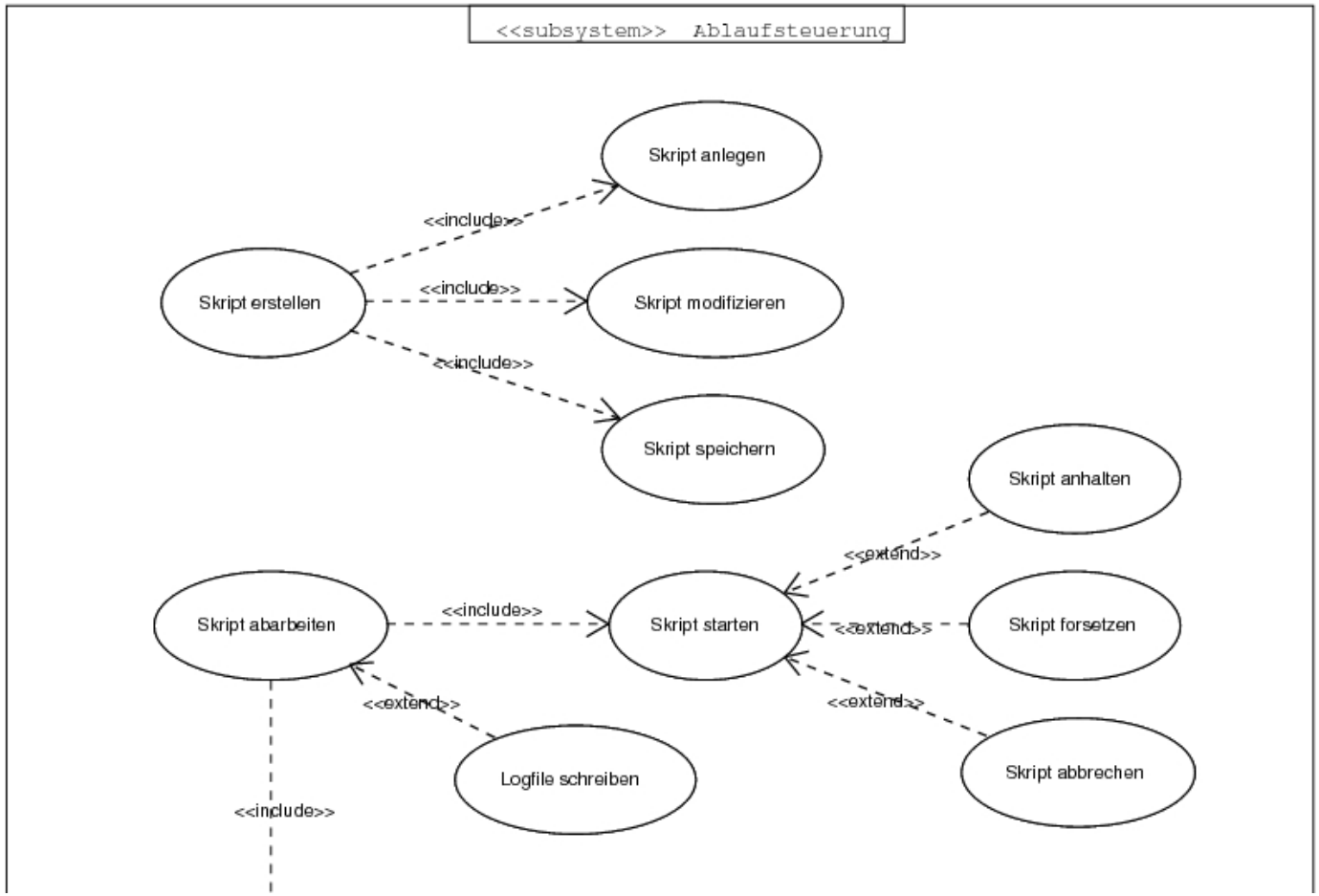
Skript in neuer Skriptsprache SPEX

```
int i = 1
int pos = 0
double x = 10.567
// absolute move theta
if (i > 0)
{
    mv theta pos
}
```

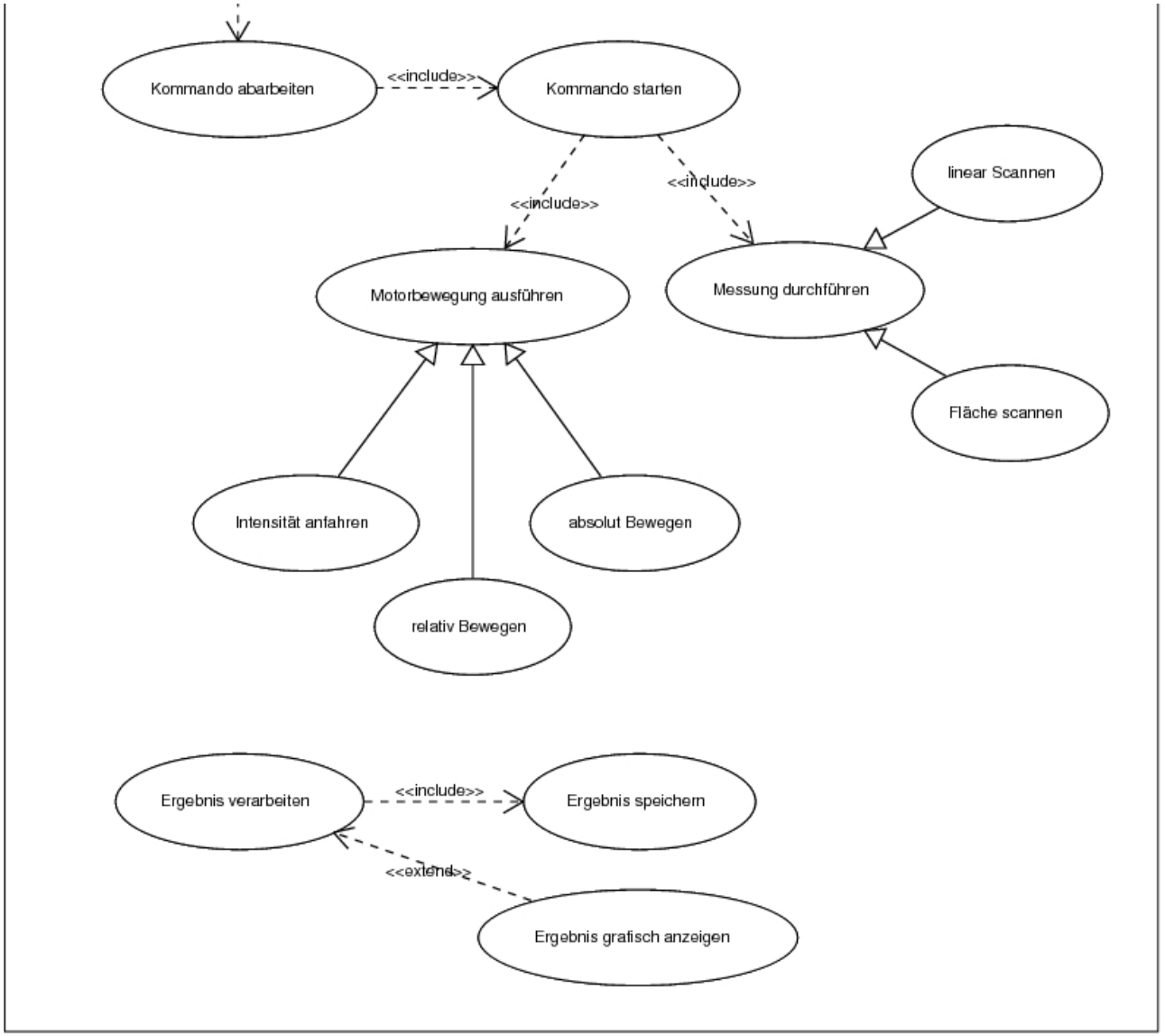
allgemeines Anwendungsfalldiagramm



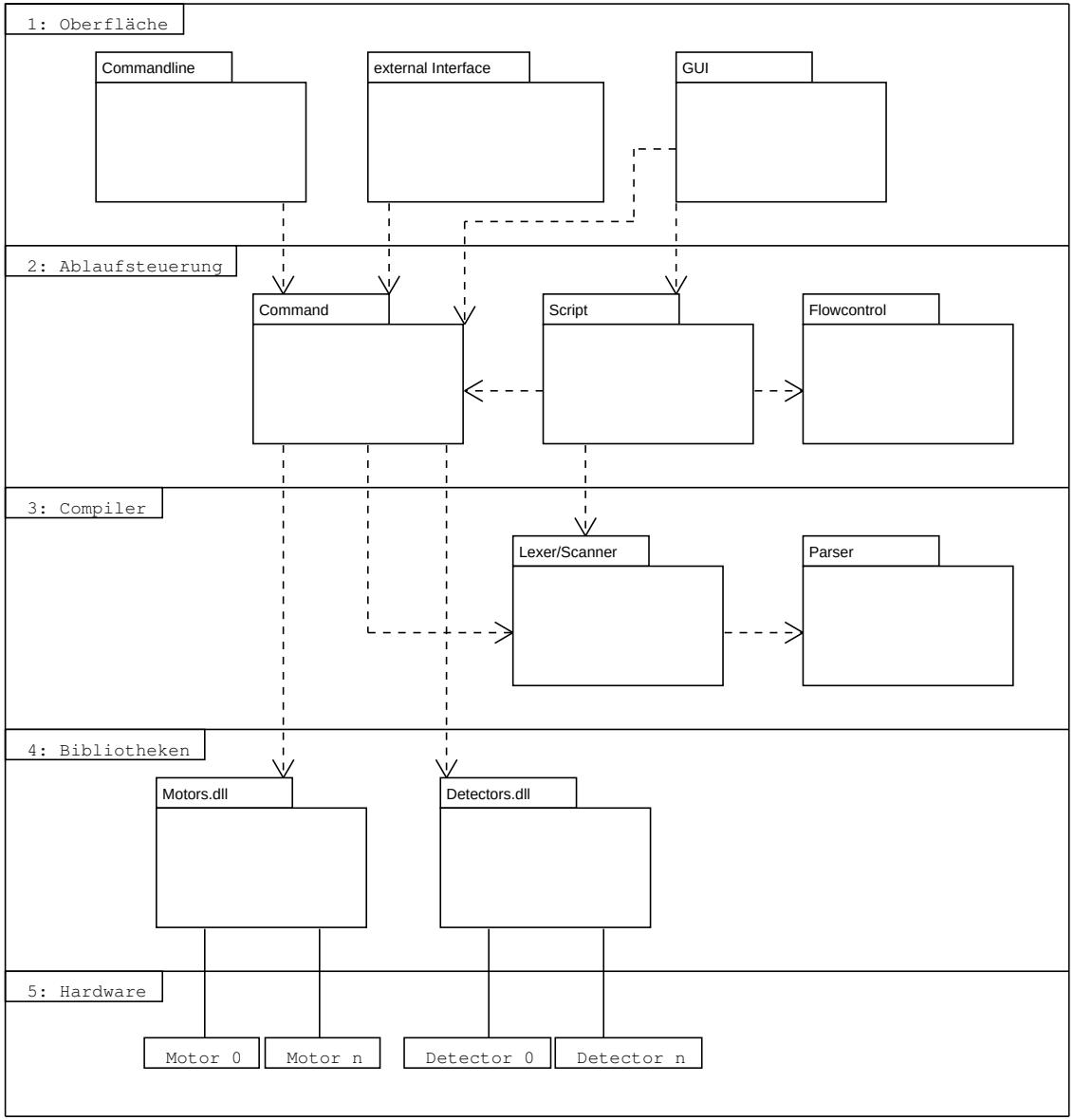
detailliertes Anwendungsfalldiagramm (1)



detailliertes Anwendungsfalldiagramm (2)



Schichtenarchitektur



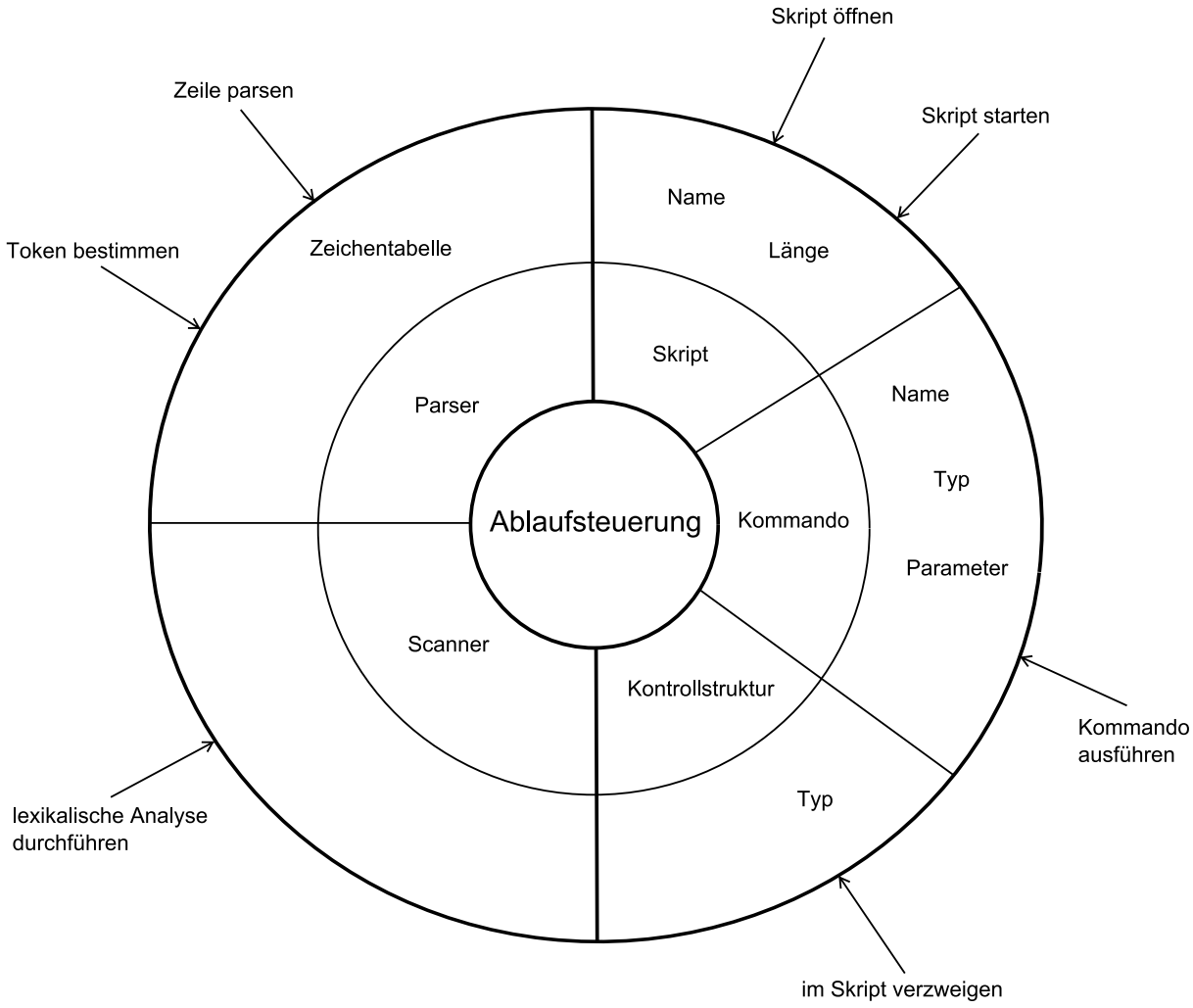
Detektoren C-Schnittstelle (c_layer.h)

- WINAPI dGetExposureValues(float&, DWORD&, float&) : BOOL
- WINAPI dlGetDevice(void) : int
- WINAPI dlGetIdByName(TDeviceType) : int
- WINAPI dlGetInstance(void) : HINSTANCE
- WINAPI dlGetVersion(void) : LPCSTR
- WINAPI dlIsDeviceValid(TDeviceType) : BOOL
- WINAPI dlParsingDevice(LPSTR) : TDeviceType
- WINAPI dlSetDevice(int) : BOOL
- WINAPI dMeasureStart(void) : int
- WINAPI dMeasureStop(void) : int
- WINAPI dSetExposureValues(float, DWORD, float) : BOOL
- WINAPI GetCounterListPtr(void) : LPDList
- WINAPI GetIntensity_A913(void) : float
- WINAPI GetIntensity_SCS(void) : float
- WINAPI GetIntensity_TDC(void) : float
- WINAPI InitializeCountersDLL(void) : BOOL
- WINAPI InitializeTDC_Event(float, HWND) : BOOL
- CALLBACK InquireIntensity_A913(UINT, UINT, DWORD, DWORD, DWORD) : void
- CALLBACK InquireIntensity_SCS(UINT, UINT, DWORD, DWORD, DWORD) : void
- CALLBACK InquireIntensity_TDC(UINT, UINT, DWORD, DWORD, DWORD) : void

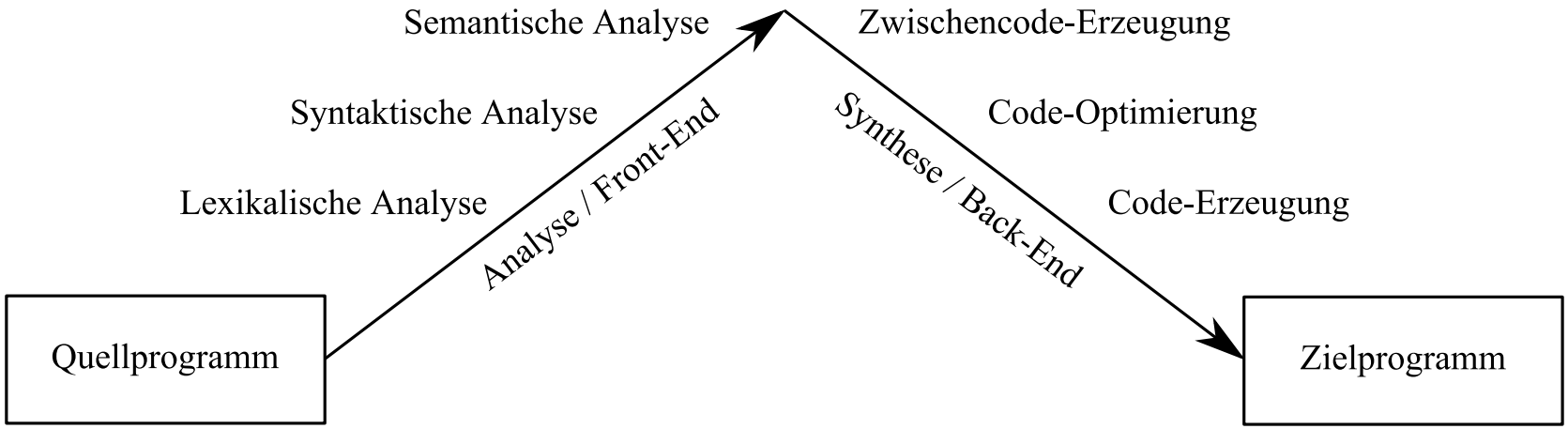
Motoren C-Schnittstelle

- mActivateDrive
- mExecuteCmd
- mGetAxisName
- mGetAxisUnit
- mGetDF
- mGetDistanceProcess
- mGetDistance
- mGetMoveFinishIdx
- mGetMoveScan
- mGetSF
- mGetScanSize
- mGetUnitType
- mGetValue
- ...
- mMoveByDistance
- mMoveToDistance
- mSetValue
- mStartMoveScan
- mStopDrive
- mlGetAxisNumber
- ...

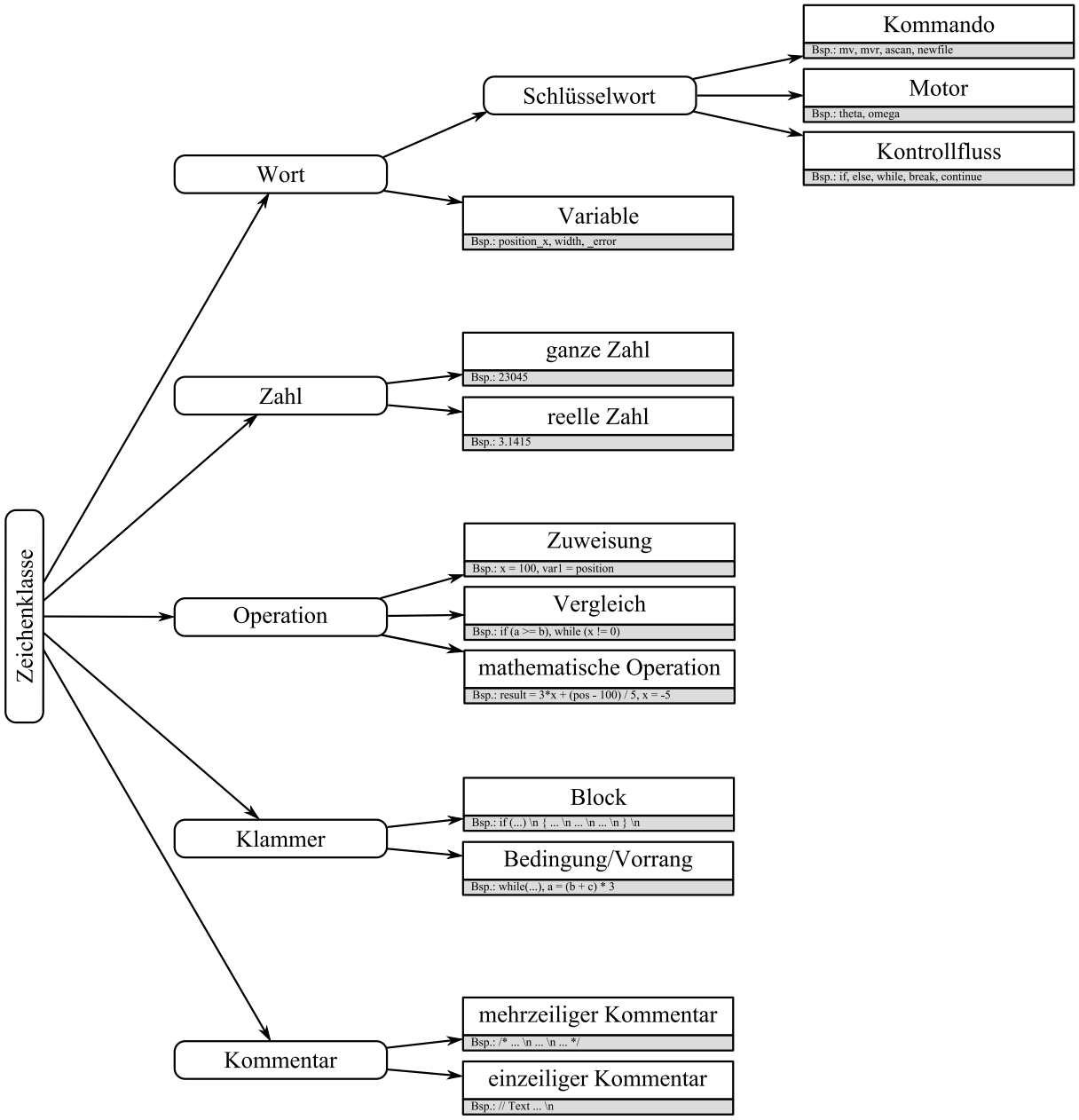
Kernfunktionalität



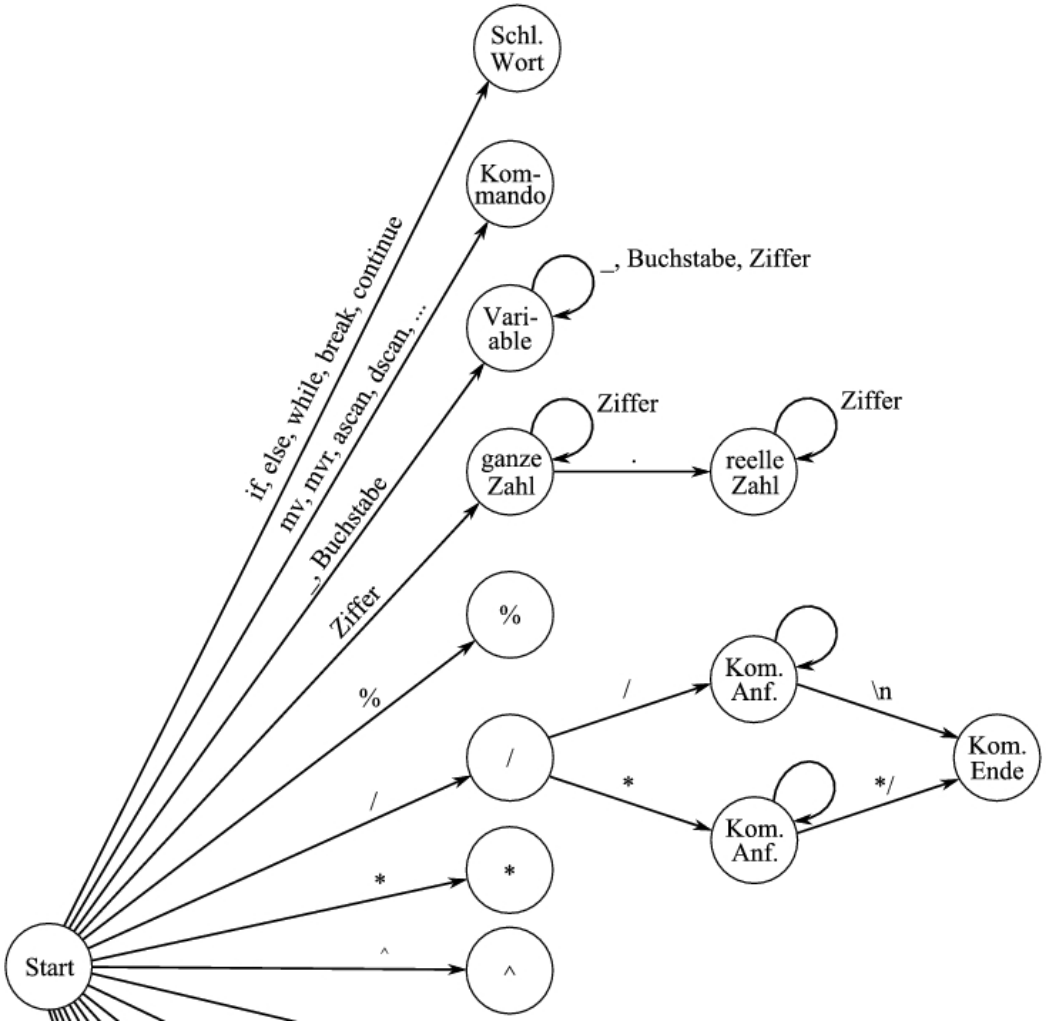
Compiler / Interpreter



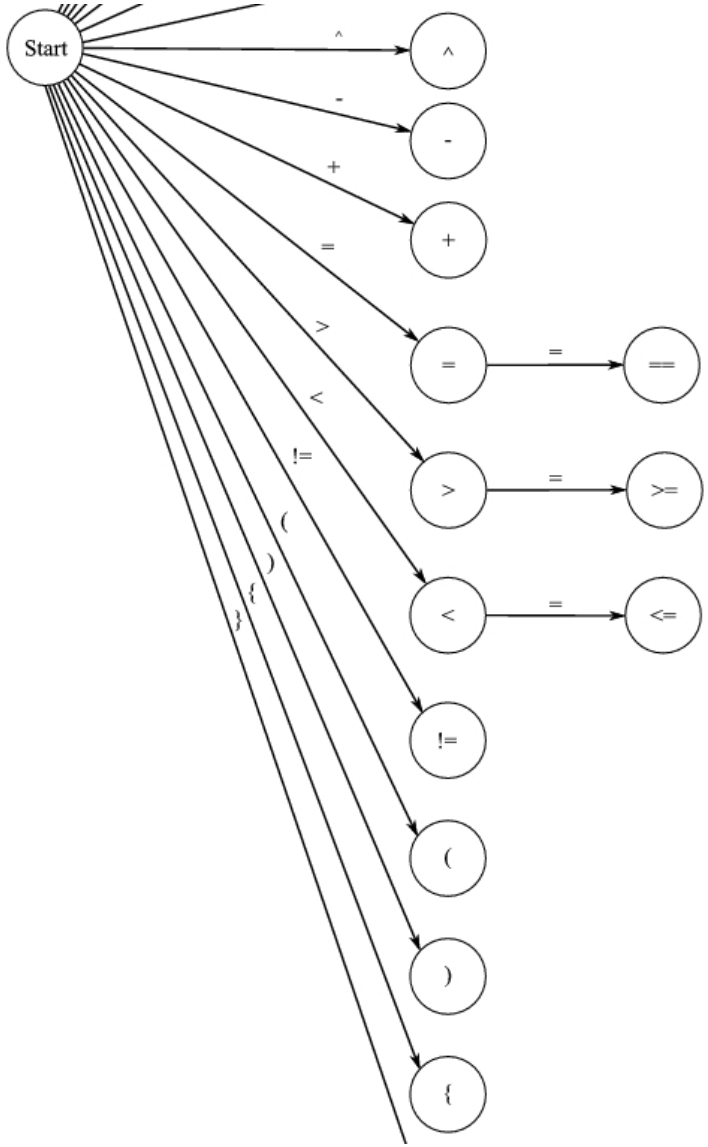
Scanner - Klassifikation der Zeichen



Scanner - Token erkennen (1)



Scanner - Token erkennen (2)



Scanner - erzeugte Dateien

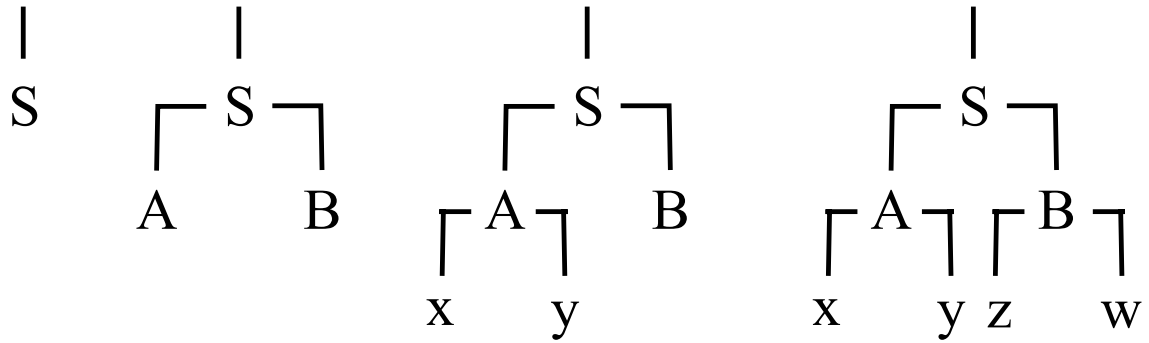
- error.log (Fehlermeldungen und Warnungen)
- symbol.log (erkannte Symbole des Skriptes)
- statistic.log (statistische Auswertung/Zählung)
- metric.log (einfache Metriken des Skripts)

Metrik	Kennung	Bewertung		
		grün	gelb	rot
Lines Of Code	LOC	< 40	< 70	$>= 70$
Lines Of Code of Commands	LOCC	< 20	< 30	$>= 30$
Lines Of Code of Empty lines	LOCE	< 5	< 10	$>= 10$
Lines Of Code of Declaration	LOCD	< 5	< 10	$>= 10$
Number Of Comment lines	NOC	$>= 5$	> 0	$== 0$
Number Of 1-line Comments	1NOC	-	-	-
Number Of more-line Comments	xNOC	-	-	-
True Comment Ratio (=NOC/LOC)	TCR	$> 10\%$	$> 5\%$	$<= 5\%$
Number Of mathematical Operations	NOO	< 20	< 40	$>= 40$
Maximum Of Block depth	MOB	< 3	< 4	$>= 4$

Parser - Satzanalyse

- orientiert sich an C-Struktur
- Anlegen von int und double Variablen (Verwaltung in der Systemtabelle durch lex und yacc)
- Kontrollstrukturen

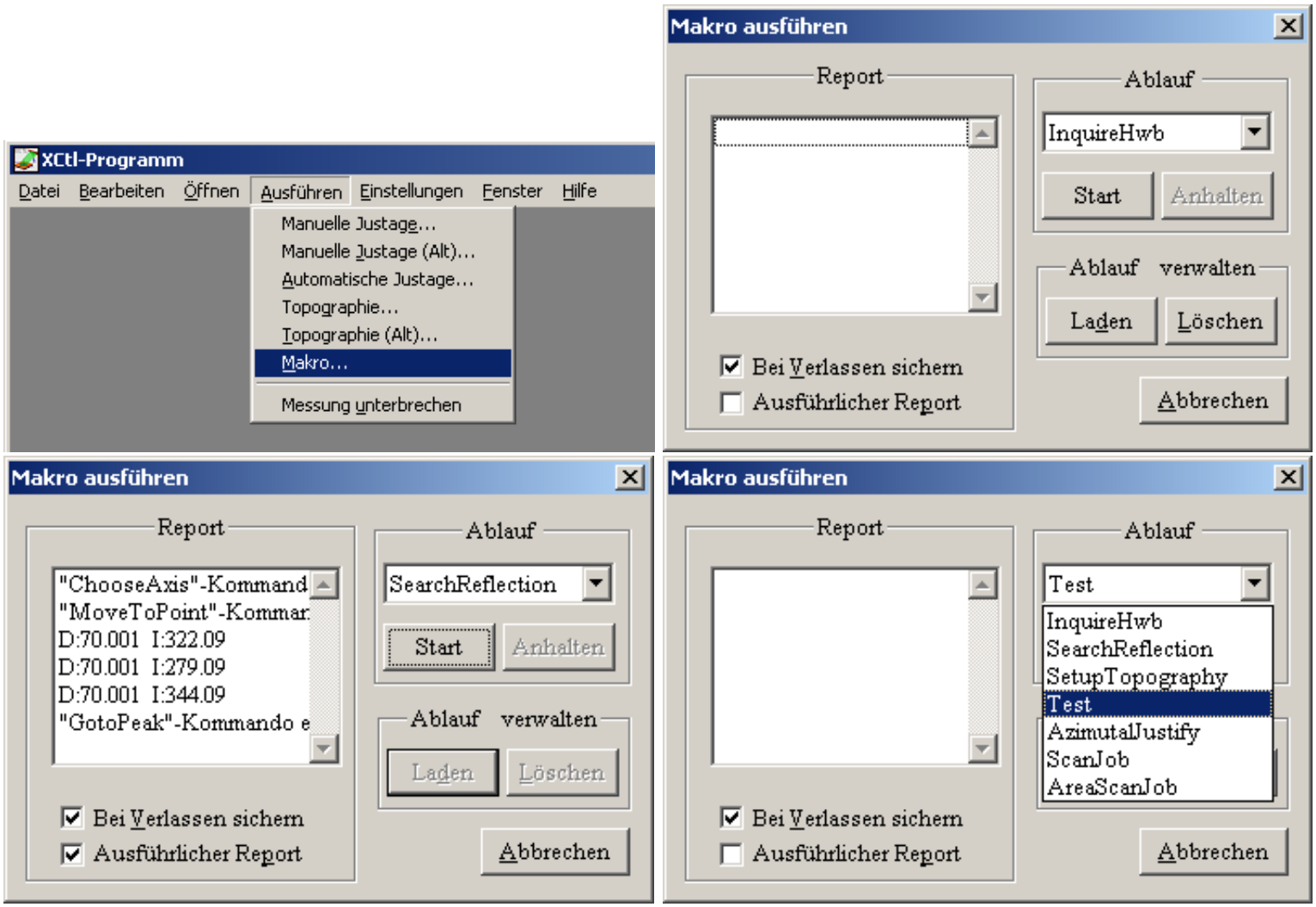
Beispielhafte Satzanalyse (Parsetree):



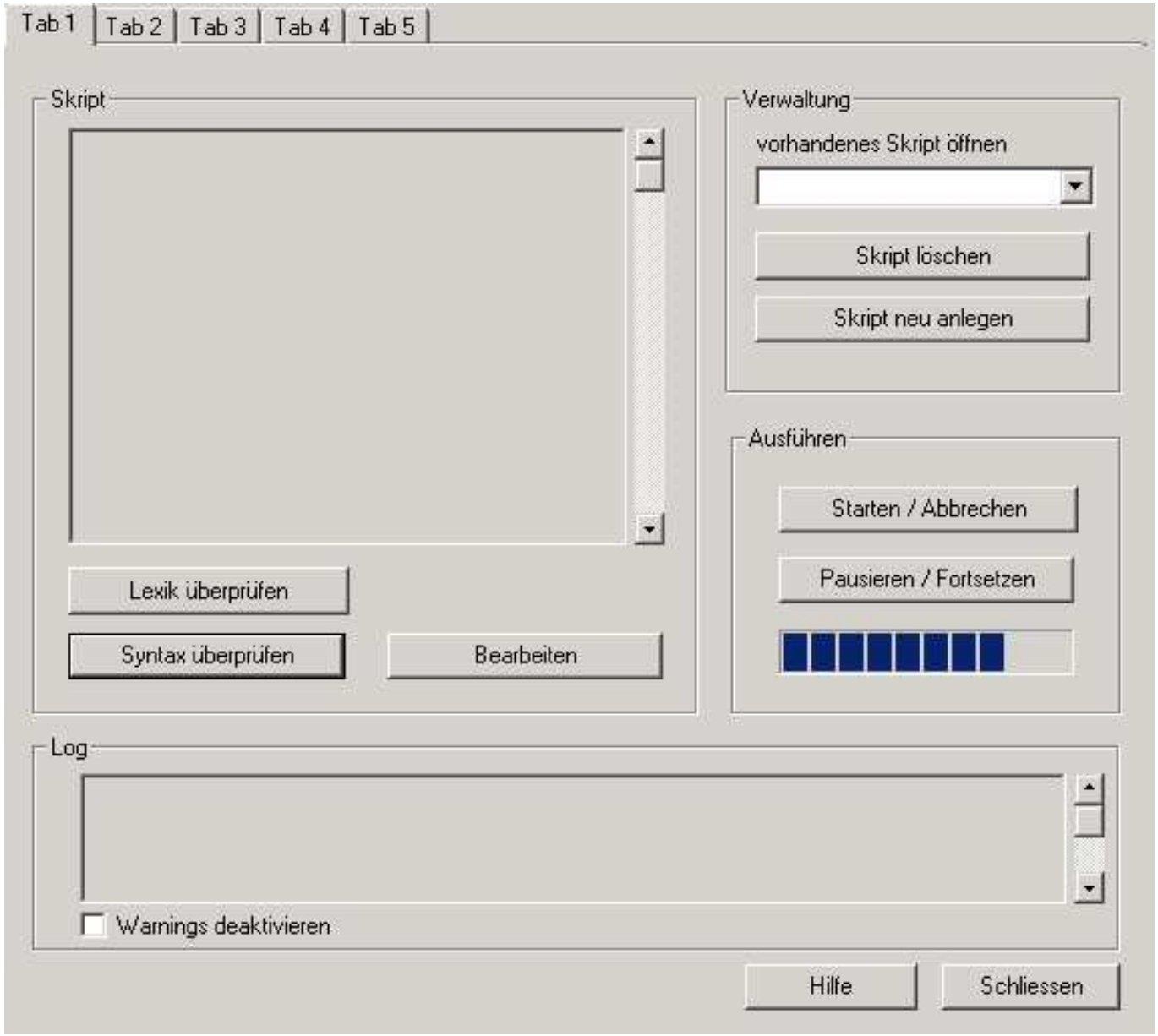
Instrumentierter Code

- Baum zur internen Repräsentation des Skript-Quellcodes
- wird vom Parser generiert
- Ausführung des Skriptes, und somit Abarbeitung des instrumentierten Codes (Baum), durch den Nutzer
- Verzweigungen werden durch Sprünge innerhalb des Baums realisiert
- zu jeder Skriptzeile existiert ein Eintrag im Baum
- Zugriff/Identifikation über Zeilennummer
- bei linearer Programmfolge normales Durchlaufen des Baumes
- bei Sprüngen (z.B. while break continue) Auswahl des entsprechenden Baumelementes

Oberfläche - altes System



Oberfläche - neues System



Vielen Dank!