

HOCHSCHULE BREMEN

BACHELORARBEIT

EXPOSÉ

Automatisierung mit Hilfe von Macros

Author:

Roland JÄGER
360956

31. Januar 2016

Inhaltsverzeichnis

1	Einleitung	2
2	Problemfeld	2
3	Lösungsansatz	5
4	Konkrete Aufgaben	6
5	Arbeitsumfeld	7
5.1	Literatur	7
5.2	Software	7
6	Planung	8
6.1	Wann	8
6.2	Wo	8
6.3	Arbeitspakete	8
6.4	Meilensteine	9
7	Gliederung der Arbeit	9
8	Personen	11
8.1	Erster Gutachter	11
8.2	Zweiter Gutachter	11
8.3	Ansprechpartner	11
8.4	Student	11
9	Unterschriften	12

1 Einleitung

Die Einführung von Automatisierung in ein Softwaresystem ist vergleichbar mit den Maschinen die in der industrielle Revolution auftauchten. Anstelle, dass Menschen arbeiten müssen, um ein gewünschtes Ergebnis zu bekommen, drücken sie auf einen Knopf und ein anderes System nimmt ihnen die aufwändige Arbeit ab. Dies führt dazu, dass Produkte schneller, mit weniger Arbeitsaufwand erstellt werden können. Zudem ist die entstehende Qualität immer auf einem gleichbleibenden Level und hängt nicht von dem Befinden der Arbeiter ab.

Die P3-group arbeitet mit Airbus um Lösungen für den Flugzeugbau zu entwickeln. Dieser Markt ist eng umkämpft, wodurch minimale Gewinne einen großen Unterschied machen können. Ein Feld welches seit Jahren immer weiter durch wissenschaftliche und technische Durchbrüche optimiert wird, sind die menschlichen Ressourcen. Automatisierung sorgt dafür, dass sich wiederholende Arbeitsabläufe – aus der Sicht des Nutzers – zu einem einzigen Schritt werden und so Zeit sparen.

Makros sind die Fließbänder der digitalen Welt und diese Arbeit beschäftigt sich mit der Entwicklung eines Makro Systems bzw. Sprache.

Makros werden durch die Verbindung kleinerer Bausteine erstellt. Diese können andere Makros oder Befehle, die die Anwendungsumgebung bereitstellt, sein. Dies ist mit einem Fließband in der Autoindustrie zu vergleichen. Jede Station ist genau für eine Aufgabe zuständig und kümmert sich um nichts anderes.

Die Makrosprache ist ein Baukasten, mit dem Makros erstellt – Fließbänder für spezielle Aufgaben erzeugt werden können.

2 Problemfeld

Makros setzen auf dem Command-Pattern¹ auf und führen durch ihre Modularität und dynamischer Natur² unter anderem dazu, dass Endbenutzer das Verhalten und sowie Potenzial der Software beeinflussen können.

Softwaresysteme haben oft das Problem, dass sie mit einigen zentralen Features anfangen, die fest definiert werden (sollten), bevor ein Vertrag geschlossen wird. Für weitere Funktionalität, die über die Vereinbarungen im Vertrag hinausgehen, muss der Vertrag

¹ Das Command-Pattern sorgt dafür, dass die Details der Implementierung dem Client(Nutzer z.B. eine Schaltfläche) nicht bekannt sein müssen, da dieser nur einen Invoker nutzt, welcher das Spezielle Command kennt, in dem die konkrete Implementation aufgerufen wird. Durch diese mehrfache Kapselung können viele Probleme in der GUI generisch gelöst werden, was die Komplexität der Anwendung verringert. Außerdem lässt sich die Chronik der Applikation zum Zurück- und Wiederherstellen sehr leicht implementieren.

² Dynamische Programmiersprachen werden erst zur Laufzeit interpretiert. Eine prominentes Beispiel wäre JavaScript.

erweitert werden. Wenn der Vertrag erfüllt ist, und im Anschluss weitere Wünsche aufkommen, muss ein weiterer Vertrag aufgesetzt werden und die vorher gelieferte Software muss angepasst, gegebenenfalls erweitert werden. Dies kann zur Folge haben, dass große Teile der Software umgeschrieben werden müssen, oder sogar, dass die Architektur der gesamten Anwendung verändert werden muss.

Wenn frühzeitig ein Makrosystem/-sprache und ein entsprechendes Erweiterungskonzept für Module bzw. Plugins eingeführt wird, ist die Wahrscheinlichkeit, dass der Kern der Applikation angefasst werden muss, wesentlich geringer. Durch diese Kombination, kann einfach ein weiteres Modul geladen werden, was die neuen Grundbausteine der Applikation hinzufügt. Diese können dann in einem neuen, oder angepassten Makro genutzt werden, um den Wunsch der Kunden zu erfüllen. Im Falle, dass es keine neuen Grundbausteine bedarf, reicht es sogar nur ein Makro zu liefern. Die Vorteile dieser Methode sind, dass – wenn man davon ausgeht, dass die benutzen Makros und Grundbausteine fehlerfrei durch ausreichendes testen der Software sind – keine neuen Bugs in den Kern der Software eingeführt werden können und somit immens zu der Stabilität der Software beigetragen wird. Ein weiterer Vorteil ist, dass die Makros mit wesentlich weniger Aufwand entwickelt werden können, da sie sich auf einem höheren Level befinden. Für Kunden ist eine nutzbare Makrosprache auch interessant, da sie zum Teil, durch das hausinterne Personal Anforderungen an die Software realisieren können, ohne den langen Weg über eine Firma zu gehen. Dies bedeutet auch, dass die Software eine bessere Chance hat, den Tod der Herstellerfirma überleben.

Die P3-group ist daran interessiert, dass sie ihren Kunden Lösungen schnell und in hoher Qualität anbieten kann. Um dies zu erreichen arbeiten sie daran, dass alle Softwaresysteme – die von ihnen angeboten werden – Automatisierung über Makros unterstützen. Wirtschaftlich rentieren sich die Makros dadurch, dass sie von den Firmen gemietet und nicht nur einmal verkauft werden. Zum Beispiel, dass alle Flugzeugteile ausgewählt werden und dann deren Gewicht ermittelt wird. Doch manchmal sollen nur spezielle Teile aus einem bestimmten Werkstoff zusammen gezählt werden. Anstelle, dass hier eine sehr Komplexe Suchfunktion entwickelt wurde, können hier zwei Makros zum Einsatz kommen, die jeweils eine Aufgabe erfüllen und somit für den Benutzer sicher zu handhaben sind.

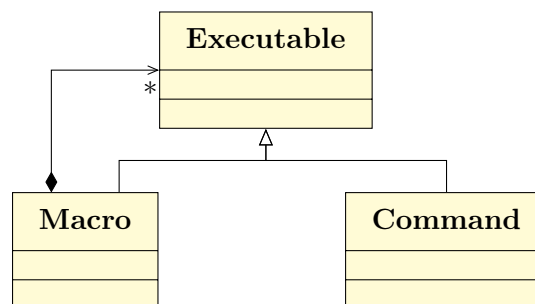


Abbildung 1: Erben von **Command** sind die Grundbausteine, Instanzen von **Macro** sind Makros, die **Executables** ausführen können, also andere Makros oder Grundbausteine.

Die Herausforderung ein Makrosystem/-sprache in C++ zu implementieren fängt dann an, wenn man von den Makros will, dass diese nicht nur hintereinander abgearbeitet werden, ohne dass sie wissen, dass andere Makros vor ihnen bzw. nach ihnen ausgeführt werden.

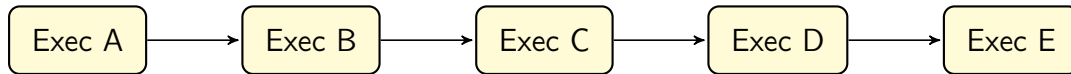


Abbildung 2: Sequenzielles abarbeiten von Executables.

Der erste Schritt zu einer Herausforderung – und zu einer nützlichen Implementation – ist Logik. Hierbei biete man an, dass der Makro Entwickler durch Rückgabewerte aus Makros entscheiden kann welche weiteren Makros er ausführen möchte.

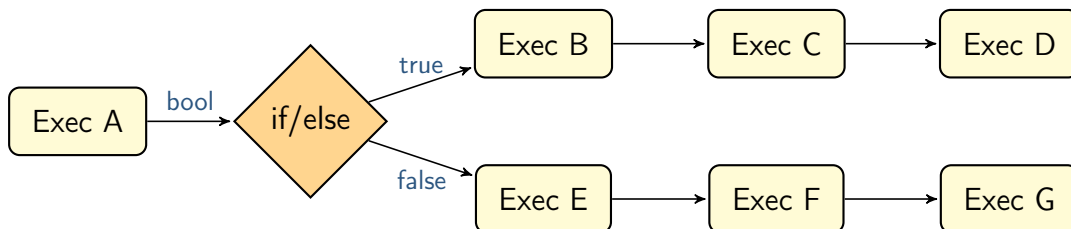


Abbildung 3: Logische Ausdrücke um bedingte Anweisungen zuzulassen.

Obwohl man mit solchen Makros schon einige Probleme lösen kann, ist es nicht das, was man zur Verfügung haben will, wenn man mit objektorientierten Sprachen arbeitet bzw. mehr als ein 'ja' oder nein 'nein' braucht. Was ein Makrosystem/-sprache anbieten muss ist, dass Instanzen von verschiedene Klassen/Typen zurückgegeben und beliebig viele Parameter (unterschiedlicher Klassen/Typen) dem Makro mitgegeben werden können. Leider sind gerade diese Punkte ein Problem in C++, da C++ keine Reflexion unterstützt. Dies ist – da es die Zeit, die für die Bachelorarbeit bereitsteht, mehrfach sprengt – schon durch eine Implementation von dem Command-Pattern von mir gelöst worden.³ Somit kommt in diesem Schritt 'nur' noch hinzu, dass es Schleifen geben kann und Rückgabewerte in Variablen speicherbar sind.

³ Das Command-Pattern wurde mit Hilfe des `any`⁴ sowie `optional`⁵ Types erstellt, womit die `Executable` Klasse ein Interface bereitstellt, mit dem Argumente jeden Types übergeben und zurückgegeben werden können.

⁴ <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3804.html>

⁵ <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3793.html>

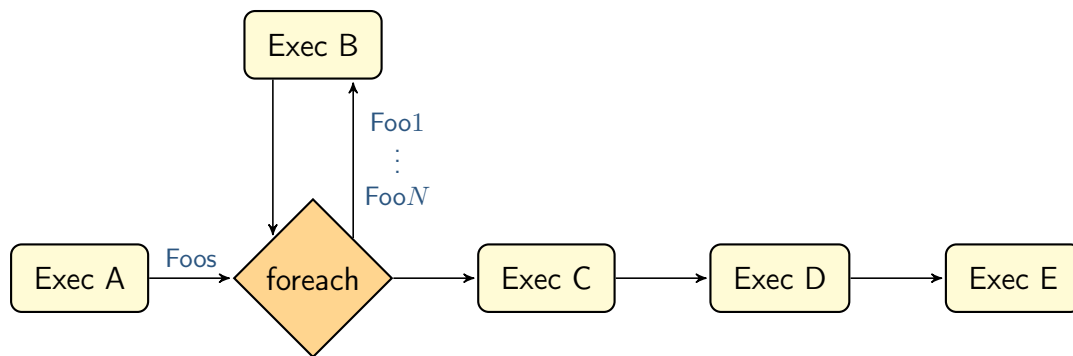


Abbildung 4: Schleife, die Anweisungen für ein Element aus der Liste aufrufen.

Letzt endlich kann man also sagen, dass ein solches Makrosystem/-sprache eine Programmiersprache mit Interpreter ist, deren Laufzeitumgebung ein anderes Softwaresystem ist.

3 Lösungsansatz

Der Makroansatz besteht aus inkrementell kompletteren Lösungen. Diese Lösungen sind in drei Level aufteilbar. Das erste Level beschäftigt sich hauptsächlich mit dem Parser⁶/Tokenizer⁷ für den Interpreter⁸ und soll Makros ausschließlich sequenziell ausführen können. Das zweite Level befasst sich mit der Erweiterung des Interpreters sowie der Einführung oder Implementierung von Argumenten und Rückgabewerten. Das dritte und letzte Level beschäftigt sich damit die Implementierungen von Makros und Interpreter weiter auszunutzen, um Schleifen von `while` bis `for-each` zu implementieren.

Bevor mit der ersten Iteration begonnen werden kann, muss recherchiert werden, was schon bei ähnlichen Problemen entwickelt wurde, und eine Anforderungsanalyse durchgeführt werden. Auf diesen beiden Ergebnissen wird dann eine Subarchitektur entwickelt, die für alle Level gültig ist und als Grundlage für die Implementierung dienen wird.

Der Parser für das erste Level bedarf einen Syntax den er parsen kann. Dieser muss für die späteren Iterationen ausreichend flexibel sein. Höchst wahrscheinlich wird der Parser den Makrostring in einen abstrakter Syntaxbaum (AST)⁹ überführen. Der Interpreter

⁶ Ein Parser ist eine Softwarekomponente, die einen Text analysieren und in ein Format bringen, welches von dem Rest der Software verstanden werden kann.

⁷ Tokenizer sind ein Bestandteil des parsens, Tokenizer zerteilen Text in sinnvolle Segmente, die von Parsern weiterverarbeitet werden können. Oft werden Tokenizer in kleineren Projekten direkt in den Parser integriert und werden daher selten extra betrachtet.

⁸ Interpreter sind Softwarekomponenten, die Befehle ausführen, ohne den Quelltext vorher zu kompilieren. Interpreter machen sich Parser zu nutzen, um den Quelltext in ein Format zu bringen, welches sie nutzen können um Befehle auszuführen.

⁹ *Abstract syntax tree* ist eine digitale Darstellung einer Programmiersprache. Diese wird von den Softwarekomponenten nach dem Parser genutzt und sorgt dafür, dass diese den Syntax der eigentlichen Sprache nicht kennen müssen und somit von Veränderungen größtenteils geschützt sind.

muss in dieser Iteration nicht viel mehr können als Makro nach Makro aufzurufen und ist somit eine leichte Aufgabe.

Im zweiten Level wird der Interpreter, durch die Einführung der bedingten Anweisungen, sowie von Argumenten und Rückgabewerte der Makros erweitert. Die Änderungen an dem Parser sollten hier nur noch minimal sein, da dieser nur weitere Tokens erkennen und in den AST überführen muss.

Die Schleifen sowie Variablen Deklarationen bauen auf die vorherigen Stufen auf und sollten bei einer perfekten Ausführung derer, mit wenig Aufwand zu implementieren sein. Sollte allerdings bei der Planung bzw. der Architektur ein Detail übersehen worden sein, kann dies zu größeren Änderungen führen.

4 Konkrete Aufgaben

Die in der Bachelor-Arbeit durchzuführenden Aufgaben sind:

- Anforderungsanalyse:
 - Was muss die Makrosystem/-sprache mindestens leisten um von Nutzen zu sein?
 - Machbarkeitsanalyse – Was gibt C++ her?
- Subarchitektur für alle Komponenten, die mit der Arbeit zu tun haben.
 - AST
 - Interpreter
 - Makros
 - Parser
 - Tokenizer
- Entwicklung eines Syntax für die Makrosystem/-sprache.
- Eine Referenz Implementierung von der Makrosystem/-sprache.
- Die Ergebnisse der Arbeit niederschreiben.

5 Arbeitsumfeld

5.1 Literatur

Literatur

cppreference.com (2015). URL: <http://en.cppreference.com/w/cpp> (besucht am 29.12.2015).

The C++ Standards Committee (2016). URL: <http://www.open-std.org/JTC1/SC22/WG21/> (besucht am 17.01.2016).

5.2 Software

- Clang¹⁰ als C++11/14 Compiler
 - Address Sanitizer¹¹ um folgende Fehler zu finden:
 - * “Out-of-bounds accesses to heap, stack and globals”
 - * “Use-after-free”
 - * “Use-after-return (to some extent)”
 - * “Double-free, invalid free”
 - * “Memory leaks (experimental)”
 - Thread Sanitizer¹² um folgende Fehler zu finden:
 - * data races
 - * mutex lock Reihenfolge (potenzielle deadlocks)
- Git¹³ als Versionsverwaltung
- CMake¹⁴ als Build-System-Generator
- Latex¹⁵ für Text der kein Code ist und Vektorgrafiken
- Sublime Text 3¹⁶ mit Plugins von der Package Control¹⁷ als Editor/IDE

¹⁰ <http://clang.llvm.org/>

¹¹ <http://clang.llvm.org/docs/AddressSanitizer.html>

¹² <http://clang.llvm.org/docs/ThreadSanitizer.html>

¹³ <http://git-scm.com/>

¹⁴ <https://cmake.org/>

¹⁵ <http://www.latex-project.org/>

¹⁶ <http://www.sublimetext.com/3>

¹⁷ <https://packagecontrol.io/>

- [Arch](#)¹⁸ und [Ubuntu](#)¹⁹ als Betriebssystem
- [Inkscape](#)²⁰ und [Visual Paradigm](#)²¹ für Grafiken
- P3-group interne Software als Arbeitsgrundlage

6 Planung

6.1 Wann

März bis Juni 2015

6.2 Wo

P3 engineering GmbH
Flughafenallee 26/28
28199 Bremen
www.p3-group.com

6.3 Arbeitspakete

- Recherche (ca.5 Tage)
- Anforderungsanalyse (ca.5 Tage)
- Konzeption (ca.5 Tage)
 - Level 1 – Abarbeiten von Macros
 - Level 2 – Logik / primitive Returnwerte (ca. $\frac{1}{2}$ Woche)
 - Level 3 – Komplexe Returnwerte (ca. $\frac{1}{2}$ Woche)
- Implementierung (ca.3 Woche)
 - Level 1 – Abarbeiten von Macros (ca.2 Tage)
 - Level 2 – Logik / primitive Returnwerte (ca.1 Woche)
 - Level 3 – Komplexe Returnwerte (ca. $1\frac{1}{2}$ Wochen)
- Dokumentation (ca.3 Tage)

¹⁸ <https://www.archlinux.org/>

¹⁹ <http://www.ubuntu.com/>

²⁰ <https://inkscape.org/en/>

²¹ <http://www.visual-paradigm.com/>

- Verfassen der Bachelor-Thesis (ca.2 Woche)
 - Einleitung + Anforderungsanalyse (ca. 1 Woche, ab 22. März)
 - * Allgemeines
 - * Kapitel 1: Einleitung
 - * Kapitel 2: Anforderungsanalyse
 - * Kapitel 3: Grundlagen und alternative Lösungen
 - Hauptteil (ca.1 $\frac{1}{2}$ Wochen, ab 12. April)
 - * Kapitel 4: Konzeption
 - * Kapitel 5: Exemplarische Realisation
 - Schlussteil (ca.3 Tage, ab 26. April)
 - * Kapitel 6: Evaluation
 - * Kapitel 7: Zusammenfassung und Ausblick

6.4 Meilensteine

	Abschluss	Beginn
1. März:		Recherche & Anforderungsanalyse
8. März:	Recherche & Anforderungsanalyse	Konzeption
22. März:	Konzeption	Implementierung & Erster schriftliche Teil
12. April:	Implementierung & Erster schriftliche Teil	Dokumentation & Zweiter schriftliche Teil
26. April:	Dokumentation & Zweiter schriftliche Teil	Dritter schriftliche Teil
2. Mai:	Dritter schriftliche Teil	Korrektur, Binden der DA etc.
6. Mai:	Abgabe der Arbeit	

7 Gliederung der Arbeit

Allgemeines

Eidesstattliche Erklärung

Danksagung

Kapitel 1: Einleitung

- 1.1. Problemfeld
- 1.2. Ziele der Arbeit
- 1.3. Hintergründe und Entstehung des Themas
- 1.4. Struktur der Arbeit, wesentliche Inhalte der Kapitel

Kapitel 2: Anforderungsanalyse

- 2.1. Diskussion des Problemfeldes
- 2.2. Angestrebte Lösung

Kapitel 3: Grundlagen und alternative Lösungen

- 3.1. *Make or Buy*
 - 3.1. *Nagios*
 - 3.2. *ServerGuard24*
 - 3.3. *PocketDBA*
- 3.2. *Eigenentwicklung*
 - 3.1. *Vorteile einer Eigenentwicklung*
 - 3.2. *Architektur*
 - 3.3. *Mobile Kommunikation*
 - 3.4. *Programmiersprachen*
 - 3.5. *Sicherheitsaspekte*

Kapitel 4: Konzeption

- 4.1. *Client-Server-Architektur*
- 4.2. *HTTPS-Server*
- 4.3. *Mobiler Client*
- 4.4. *Webclient*

Kapitel 5: Exemplarische Realisation

- 5.1. *Systemvoraussetzungen*
- 5.2. *Hard- und Software*
- 5.3. *HTTPS-Server*

5.4. Mobiler Client

5.5. Webclient

Kapitel 6: Evaluation

Kapitel 7: Zusammenfassung und Ausblick

Anhänge

8 Personen

8.1 Erster Gutachter

Name: Prof. Dr. Thorsten Teschke

E-Mail: thorsten.teschke@hs-bremen.de

8.2 Zweiter Gutachter

Name:

E-Mail:

8.3 Ansprechpartner

Name: Mirko Wiechmann

E-Mail: Mirko.Wiechmann@p3-group.com

Tel.: +49 421 55 83 64 300

8.4 Student

Name: Roland Jäger

Matrikelnr.: 360956

E-Mail: roland@wolfgang-jaeger.de

Tel.: +49 163 636 43 02

9 Unterschriften

Ort	Datum	Prof. Dr. Thorsten Teschke
Ort	Datum	Zweiter Gutachter
Ort	Datum	Mirko Wiechmann
Ort	Datum	Roland Jäger