Technologisches Gewerbemuseum

Höhere Lehranstalt für Informationstechnologie

DIPLOMARBEIT

Gesamtprojekt

Keks-o-bot

Betreuer

Mag. Erhard List, BSc.

DI (FH) Mag. Dr.techn. Gottfried Koppensteiner

Grafikverarbeitung & Auftragsumwandlungs Roman Gschiegl	Systemtechnik	5YHITM
Produktionsschnittstelle & Produktionsanlage Miriam Hauer	Systemtechnik	5YHITM
Zentrales Backend Christoph Miko	Systemtechnik	5YHITM

ausgeführt im Schuljahr 2016/17

Abgabevermerk:

Datum: 31.03.2017

übernommen von:

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen-, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 31.03.2017

Roman Gschiegl

Gschiegl Roman

Hauer Miriam

Hauer Miriam

Miko Christoph

Abstract

Die Vision der Industrie zielt auf die Anpassung der Fertigungsanlage für individuelle Wünsche der Kunden ab. Dabei spielen Roboter eine große Rolle. Derzeitiger Stand der Produktion ist, dass sie auf hohe Stückzahlen gleicher Produkte ausgelegt ist.
Dieses Projekt soll die Möglichkeiten und Herausforderungen einer adaptiven Fertigungsanlage für individuelle Produkte bis zur Losgröße 1 im Sinne der "Industrie 4.0" anhand eines realistischen Szenarios aufzeigen. Es möchte demonstrieren, dass die Kommunikation von einer hohen Softwareebene (einem Webshop) bis zur Fertigungsebene mit Robotern zu einem Produkt führt, dass aus den individuellen Wünschen des Kunden entstanden ist.
Es wird ein Web-Portal mit einfachem Shop-System und Benutzerverwaltung mit Login zur Verfügung gestellt, bei dem der Kunde zwischen vorgefertigten Mustern wählen kann, oder seinen eigenen Wunsch bekannt gibt. Die rein geometrischen Rohdaten aus der Verzierungsvorlage werden in neutrale Anweisungen übersetzt. Dadurch ist auch der Umstieg auf einen anderen Roboter jederzeit möglich. Das neutrale Format wird von einer Logik, die es in roboterspezifische Bewegungssätze umwandelt, spezialisiert und pflegt einen Reinigungsvorgang ein. Der Roboter verziert die Kekse nun nach Kundenwunsch.

Abstract

The vision of the industry is developing towards creating factories which are able to respond to particular customer requirements. In this case the use of robots is fundamental. At the moment factories are designed to produce large numbers of the same product model.
This project is said to show the opportunities and challenges of an adaptive plant for individual products starting with the batch size of one unit in terms of the “Industry 4.0” by means of a realistic scenario]. It shall demonstrate that the communication from a high level software layer to a plant with robots can lead to a product that is based on particular customer wishes.
A web portal with a simple shop-system and accountmanagement was implemented. The customer can select a pre-designed cookie decoration or upload one of his own. Its geometric data will then be interpreted and converted into a hardware-neutral command format so that it can be used for various robot models. Afterwards a program transforms the hardware-neutral format into a set of specific robot instructions. It also adds a cleansing process. Now the robot decorates the cookie according to the customer’s request.

Danksagungen

Während der Projektlaufzeit wurden wir von einigen großartigen Menschen und Organisationen unterstützt. Hiermit möchten wir unseren Dank speziell aussprechen an:

Prof. List alias *Keksmeister*	für die intensive Unterstützung bei persönlichen und technischen Herausforderungen, sowie die beispiellose Betreuung zu jeder Tageszeit und an jedem Wochentag.
Prof. Koppensteiner	für die motivierenden Zusprüche und den Ansporn, immer noch ein wenig mehr zu wollen. *Just do IT!*
Verein PRIA	für die Bereitstellung der Räumlichkeiten und des Equipments, das essenziell für die Projektentwicklung war. Danke auch an die Vereinsmitglieder, die uns bei der ein oder anderen technischen Tücke geholfen haben.
Abteilung HIT	dafür, dass sie durch den Unterricht bei uns das Interesse an Industrieroboter geweckt hat.
Buffetdame Carla	dafür, dass sie unseren Schlafmangel mit köstlichem Kaffee kompensiert und uns mit ihrer positiven Art aufgemuntert hat.
`Git`-Entwickler	dafür, dass sie unvorstellbar viele Lebensstunden investiert haben, um uns in den zeitkritischsten Situationen mit einem Merge-Conflict zu bereichern.

Danke!

Inhaltsverzeichnis

Autor-Kürzel: RG – Roman Gschiegl, MH – Miriam Hauer, CM – Christoph Miko

Kapitel 1

Einleitung

1.1 Motivation (MH)

Bei “keks-o-bot” handelt es sich um ein Diplomprojekt das als Abschlussarbeit in der Abteilung Informationstechnologie am TGM erarbeitet wird. Sie wird in Kooperation mit dem Verein “PRIA” (Practical Robotics Institute Austria) durchgeführt.
Behandelt wird die Konzepterstellung und prototypische Entwicklung eines vollautomatischen Systems zur Bestellung und Verzierung personalisierter Kekse mithilfe von Industrierobotern.
Wir hatten verschiedenste Motivationen, die uns dazu bewogen haben bei diesem und keinem anderen Projekt teilzuhaben. Hauptsächlich wurden wir von der Innovation der Projektidee angetrieben, da es keine anderen vergleichbaren Produkte am Markt gibt. Die Kombination aus High-Level Software, wie es eine Website beispielsweise ist, und Low-Level Industriesystemen wird im Zuge der Industrie 4.0 zwar in Zukunft öfter aufkommen, ist zum momentanen Zeitpunkt allerdings noch eine Seltenheit.
Außerdem waren wir es als Informationstechnologen stets gewohnt nur auf Bildschirme zu starren, um dann bestenfalls einen Commandline-Output oder eine Graphical User Interface (GUI) als Reaktion zu erhalten, was auf Dauer Begeisterung und Motivation immens reduzieren kann. In diesem Projektumfeld genießen wir das reale Feedback durch den Roboter.
Auch für Außenstehende, die nicht wissen was alles im Hintergrund zur Entwicklung dazu gehört, ist es einfacher nachzuvollziehen wenn sie eine tatsächliche Bewegung beziehungsweise Veränderung erkennen.
Der steigende Trend zur Industrie 4.0 und unsere zeitige Außeinandersetzung damit, verschafft uns außerdem einen Vorteil in unserer zukünftigen Laufbahn.
Zusätzlich liegen unsere Interessen verstärkt in diesem Bereich und es wird allen Teammitgliedern ein tieferer Einblick in diesen Teil der Systemtechnik ermöglicht.

... Und Kekse.

1.2 Aufgabenstellung (RG)

Es ist ein Softwarepaket zu entwickeln, mit dem ein Verzierungsdienst für Kekse aufgebaut werden kann. Die Hardware ist dabei bereits am Produktionsstandort vorhanden. Durch Übermittlung eigener Verzierungsgrafiken können Kunden ihre Bestellung personalisieren. Die Anpassung an Kundenwünsche gepaart mit der Möglichkeit kleiner Stückzahlen stellt einen Charakter der "Industrie 4.0" dar. Ziel ist es zu zeigen, dass die Auftragsannahme bis zur Ausführung in der Produktionsebene automatisiert ablaufen kann.

Ein Shopsystem mit einfach gehaltenen Funktionen soll entwickelt werden. Neben der Angabe von Rechnungs- und Versanddaten soll eine Auswahl des Kekstyps, des Designs der Verzierung, gewünschte Glasurmaterialien und die Anzahl an Keksen ermöglicht werden. Bei der Verzierung hat der Kunde die Wahl zwischen dem vom Verzierungsdienst angebotenen, vorgefertigten Keksdesign und einem eigenen Keksdesign. Das eigene Keksdesign gibt der Kunde durch Upload einer Vektorgrafik-Datei bekannt. In jedem Fall wird dem Kunden die ausgewählte oder hochgeladene Verzierung auf dem ausgewählten Kekstyp als Vorschau angezeigt. Das dient ihm zur Verifizierung, dass die Verzierung wunschgetreu positioniert und skaliert ist. Bestehen Änderungswünsche, soll ein einfacher Grafikeditor das Neupositionieren und Skalieren ermöglichen.
Dem Kunden steht die Wahl einer Bezahlungs- und Versandmethode frei, wobei diese Funktionalität im Endprodukt nur prototypisch eingebaut wird. Die angebotenen Optionen werden sehr von der Outputmenge des Verzierungsdienstes (also seiner Menge an Aufträgen), außerdem natürlich vom Standort des Kunden abhängen.
Der Bestellvorgang setzt voraus, dass der Kunde mit seinem Benutzerkonto eingeloggt ist oder dafür ein neues Konto anlegt.
Schließlich bestätigt der Kunde die Einverständnis rechtlicher Vereinbarungen und gibt seinen Auftrag auf.

Für jedes unterstützte Glasurmaterial sollen eigene Anforderungen an die Produktionsebene gestellt werden können. Dazu zählt etwa die maximal mögliche Bewegungsgeschwindigkeit des Roboterarms, bei der eine standardisierte Verzierung noch fehlerfrei (bzw. im Toleranzbereich) ausgeführt werden konnte. Außerdem werden andere Viskositäten einen anderen Pumpdruck benötigen. Besonders hilfreich wäre die Möglichkeit neue Materialien mit einem speziellen Benutzerstatus ("Bäcker") auf der Webseite einrichten zu können, ist aber ein Kann-Kriterium.

Aus dem erhaltenen Auftrag werden die relevanten Informationen entnommen, um daraus eine Menge an Roboteranweisungen ("Verzierungsrezepte") zu definieren. An dieser Stelle würde normalerweise eine Abhängigkeit von einem bestimmten Robotermodell bzw. zumindest einer Anweisungssprache des jeweiligen Roboterherstellers entstehen. Das Ziel ist aber, dass der Kunde seinen Verzierungsroboter unabhängig von den Voraussetzungen des Software-Endprodukts auswählen kann. Daraus folgt, dass ein anlagenneutrales Roboter-Anweisungsformat verwendet wird. Für die Ausführung der anlagenneutralen Roboteranweisungen muss dann eine für den Robotertyp spezifische Verbindungskomponente entwickelt werden, wobei es nicht Ziel des Endprodukts ist diese für den Kunden des Endprodukts mitzuliefern. In einer prototypischen Produktionsanlage soll der vorhandene *KUKA KR6 sixx 900 agilus* verwendet werden, um den Verzierungsvorgang prototypisch auszuführen.
Der grafische Inhalt der Keksverzierungen muss mit den Zusatzanforderungen der in der Vezierung verwendeten Materialien kombiniert werden, um die Roboterbefehle zu definieren. Die Grafik wird in ihre einzelnen grafischen Objekte geteilt und daraus für Roboter verständliche Einzelbewegungen definiert,

dazu noch für das verwendete Material die Zusatzanforderungen vermerkt.

Als optional zu entwickelnde Funktionen sind definiert: die Verwaltung von Keksen und Glasurmaterialien auf der Webseite, ein Livestream aus der Produktion für Kunden, ein einfacher Auftragsreihungs-Algorithmus.

1.3 State of the Art (MH)

1.3.1 Ausgangslage

Im Moment ist die Industrie darauf spezialisiert gleichartige Produkte in hohen Stückzahlen möglichst gewinnbringend zu produzieren.

Mit der "DIY (Do it yourself)" Welle wurde, nach eigenem Empfinden, das Bedürfnis nach Individualismus in den persönlichen Umfeldern immer merklicher spürbar. Freunde und Familie legen, vorallem bei Geschenken, immer mehr Wert auf den Touch der Einzigartigkeit. Das Angebot auf diesem Gebiet wird immer präsenter, deckt aber noch nicht alle Produkte ab.
Unter dem Begriff "Industrie 4.0" wird eine neue Generation der Industrie verstanden, die sich vorallem mit personalisierten Produkten auseinandersetzt. Weiters soll die Produktion so flexibilisiert werden, dass auf spezielle Kundenwünsche problemlos eingegangen werden kann.
Die Produktionsmengen können, bis auf die Losgrößen 1 (= ein produziertes Produkt beziehungsweise eine "Packung") reduziert werden. Außerdem müssen die verwendeten Technologien erweiterbar und anpassungsfähig sein um die verschiedenen Kundenwünsche, ohne Qualitätsverlust hinsichtlich Verarbeitung und Dauer, erfüllen zu können.

1.3.2 Ähnliche Produkte

Verzierung durch Roboter: Klötzchen für Kekse

Klötzchen für Kekse ist ein Artikel in dem mithilfe eines EV3 Lego Mindstorm Sets ein mini-Roboter zur Verzierung von Keksen selbst gebaut wird [11]. Die Überschneidung der Projekte befindet sich hier bei der automatischen beziehungsweise durch Roboter gesteuerten Verzierung von Keksen mit der Losgröße 1.
Allerdings differenzieren sie sich stark hinsichtlich der Erweiterbarkeit und der Grundidee. Während "keks-o-bot" zwar als Prototyp ebenfalls auf die Losgröße 1 ausgelegt wird, ist die Erweiterung auf andere Losgrößen angedacht und eingeplant, wohingegen dies bei "Klötzchen für Kekse" schon aufgrund des Platzmangels nicht denkbar ist.
Die Idee unterscheidet sich dahingehend, dass das dargestellte Produkt bei *Klötzchen für Kekse* nicht für den Vertrieb, sondern nur für die Heimanwendung geeignet sein soll. Bei "keks-o-bot" ist der Grundgedanke allerdings der Vertrieb des Endprodukts.

Personalisierte Produkte

Es gibt viele Anbieter im Internet für verschiedenste personalisierte Produkte. Darunter fallen zum Beispiel Unternehmen die Kleidung mit eigenen Aufdrucken in Auftrag nehmen oder solche die Bilder auf Alltagsgegenstände wie Handyhüllen, Tassen, oder Ähnliches drucken.
Diese Grundideen ähneln der des Keksverzierungs-Dienstes schon eher, wobei unklar ist ob bei der Um-

setzung ebenfalls auf automatisierte Industrieroboter, oder andere Technologien, gesetzt wird. Abgesehen von der verwendeten Technologie zur Umsetzung zielen sowohl diese Online-Dienste, als auch keks-o-bot darauf ab, individuelle Kundenbedürfnisse zu befriedigen. Dabei wird auch auf ähnliche Weise vorgegangen. Der User befindet sich in einem Online-Portal, das ihm die Möglichkeit bietet ein Motiv nach seinen Wünschen auf einen beliebigen Gegenstand zu projezieren. Danach darf er Position, Größe und Farbe nach seinen Vorstellungen anpassen. Letztenendes wird die Bestellung abgewickelt und das Produkt zu ihm nach Hause geschickt. Werbung für solche Dienste werden eher selten gesehen und wenn, dann nur in der Anfangsphase. Mittlerweile verlässt man sich auf Mundpropaganda und die eigenständige Suche der Kunden nach Anbietern, anstatt Bedürfnisse zu erzeugen. Im Lebensmittelbereich findet sich allerdings keine vergleichbare Umsetzung von Kundenwünschen.
Direkte Mitbewerber finden sich daher ebenfalls keine. Sinngemäß wäre der "Unique Selling Point (USP)" daher, dass keks-o-bot das erste Produkt dieser Art ist.

1.4 Projektmanagement (MH)

1.4.1 Verwendete Projektmanagementmethode

Für das Projekt wird die Projektmanagementmethode SCRUM verwendet. Es bietet dem Team mehr Bewegungsfreiheit wenn es zu Änderungen kommt, die heutzutage in der Technik sehr oft vorkommen. Außerdem gibt es für manche Teile des Projekts noch keine allgemeinhin akzeptierte Musterlösung, wie sie oft durch Design-Pattern oder Bibliotheken geboten wird, sondern verschiedenste Lösungsansätze die vor der endgültigen Implementierung ausgiebigst getestet werden müssen, bevor gesagt werden kann, welche gewählt werden soll. Dementsprechend ist es nicht möglich zu Beginn des Projekts zu sagen welche Wege verwendet werden sollen, wie es beim klassischen Projektmanagement nötig ist, und auch eine Machbarkeitsstudie, die in beschränktem Rahmen stattfinden muss, ermöglicht keine konkrete Entscheidung.
Darüber hinaus sieht traditionelles Management kein sogenanntes "Pair-Programming" vor. Dabei handelt es sich um eine moderne Entwicklungsmethode bei der mehrere Programmierer gleichzeitig an dem selben Modul arbeiten, wobei einer schreibt und ihm 1 - n weitere Fachkräfte über die Schulter schauen und die einzelnen Schritte im gemeinsamen Diskurs beschlossen werden. Dadurch können Nachlässigkeitsfehler und Denkfehler ausgebessert werden bevor sie implementiert werden.
Die Projektmanagementmethode wurde allerdings nicht direkt nach den SCRUM-Regeln umgesetzt. Das Daily-Standup-Meeting wurde durch wöchentliche Besprechungen ersetzt, da in einem Diplomprojekt täglich nicht so viel passieren kann wie in dem Projekt eines Unternehmens.

1.4.2 Verwendete Tools

Allgemeine Aufzeichnungen

Es wird Taiga, ein kostenloses open-source Programm, verwendet. Es bietet die für SCRUM benötigten Grundfunktionen, wie Sprint-Einteilung, User-Story-Erstellung, Point-Vergabe und Aufgaben-Zuweisung, lässt die nicht benötigten Zusatzfunktionen allerdings aus und ist insgesamt übersichtlicher gestaltet, so dass keine neuen Probleme durch unpraktische Software entstehen sollten. Jedenfalls werden alle User-Stories und Sprinteinteilungen in Taiga eingetragen. In einem übersichtlichen Burndown-Chart und einer KANBAN Ansicht kann der Fortschritt geplant und überwacht werden.

Arbeitszeiterfassung

Zeiterfassung spielt eine vordergründige Rolle in der Umsetzung eines Projekts. Sie bietet den Mitgliedern eine Übersicht über die bereits geleistete Arbeit und zeigt wie gut sich jeder selbst und seine Aufgaben einschätzen kann. Für den Product Owner und den Projekt-Betreuer ist es ebenfalls eine tolle Möglichkeit um die Fortschritte und die aufgewendete Arbeitszeit sehen und beurteilen zu können.
Aufgrund der Erfahrung die in vergangenen Projekten gesammelt wurde, wird beschlossen Toggl zu verwenden. Die Handhabung gestalltet sich sehr einfach da bloß ein Timer gestartet und beendet werden muss, was entweder online oder per App erfolgen kann. Außerdem lässt sich ein Projekt auswählen und ein Task eintragen. Zusätzlich können die Einträge für ein Projekt so exportiert werden, dass sie direkt in Taiga integrierbar sind. Ein zusätzliches, äußerst hilfreiches, Feature ist die automatische Erstellung von Statistiken, sogenannten “Reports”. Diese können entweder Aufzeichnungen des gesamten Teams oder nur einzelner Mitglieder über vor- oder selbst-definierte Zeiträume beinhalten.

Kapitel 2

Keks-o-bot: eine Übersicht

In Abbildung 2.1 wird der Aufbau des Keksobot-Softwarepaketes dargestellt. Es ist die Einteilung in getrennte, miteinander arbeitende Software-Komponenten erkennbar.

Den Komponenten sind in dieser Arbeit eigene Kapitel gewidmet, zu denen die folgende Tabelle 2.1 führt.

Komponente	**Kurzbeschreibung**	**Details in**
Web-Auftritt	Die Kunden- und Administratorschnittstelle wurde als Webauftritt konzipiert. Der Kunde wird über das Angebot informiert und er tätigt Bestellungen. Der Administrator des Keksverzierungs-Dienstes verwaltet Aufträge und das Angebot.	nicht vorhanden
zentrales Backend	Beantwortung von HTTP-Anfragen, Bestellannahme und Bestellverarbeitung. Für die Speicherung von Aufträgen in die Datenablage verantwortlich.	Kapitel 4
Datenablage	Speichert Informationen zum Kundenauftrag sowie den verfügbaren Produktkatalog und Benutzerkonten.	Abbildung 4.1
Grafikverarbeitung	Bereitet Keksverzierungen des Kunden sowie des Dienstleisters auf. Nimmt ein SVG-Dokument entgegen und wandelt es in eine an Keks-o-bot angepasste Version.	Kapitel 5
Auftrags-umwandlung	Entwickelt aus einem eingegangenen Auftrag ein roboterneutrales Anwendungsformat: das *Keksobot Robot Neutral Format*. Die Keksverzierung wird in roboterneutrale Bewegungsanweisungen gewandelt, die ausgewählten Glasurmaterialien und ihre Materialattribute vermerkt.	Kapitel 6
Produktions-schnittstelle	Erzeugt mit dem Wissen über die Produktionsanlage aus dem *Keksobot Robot Neutral Format* spezifische Roboteranweisungen, die auf die Produktionsanlage angepasst sind. Kommuniziert mit der Produktionsanlage, um die Anweisungen zu übertragen.	Kapitel 7
Produktionsanlage	Führt die übertragenen Roboteranweisungen aus. Es kommt zur Verzierung des Kekses.	Kapitel 7

Tabelle 2.1: Der Kapitel-Wegweiser zu den Keksobot-Komponenten

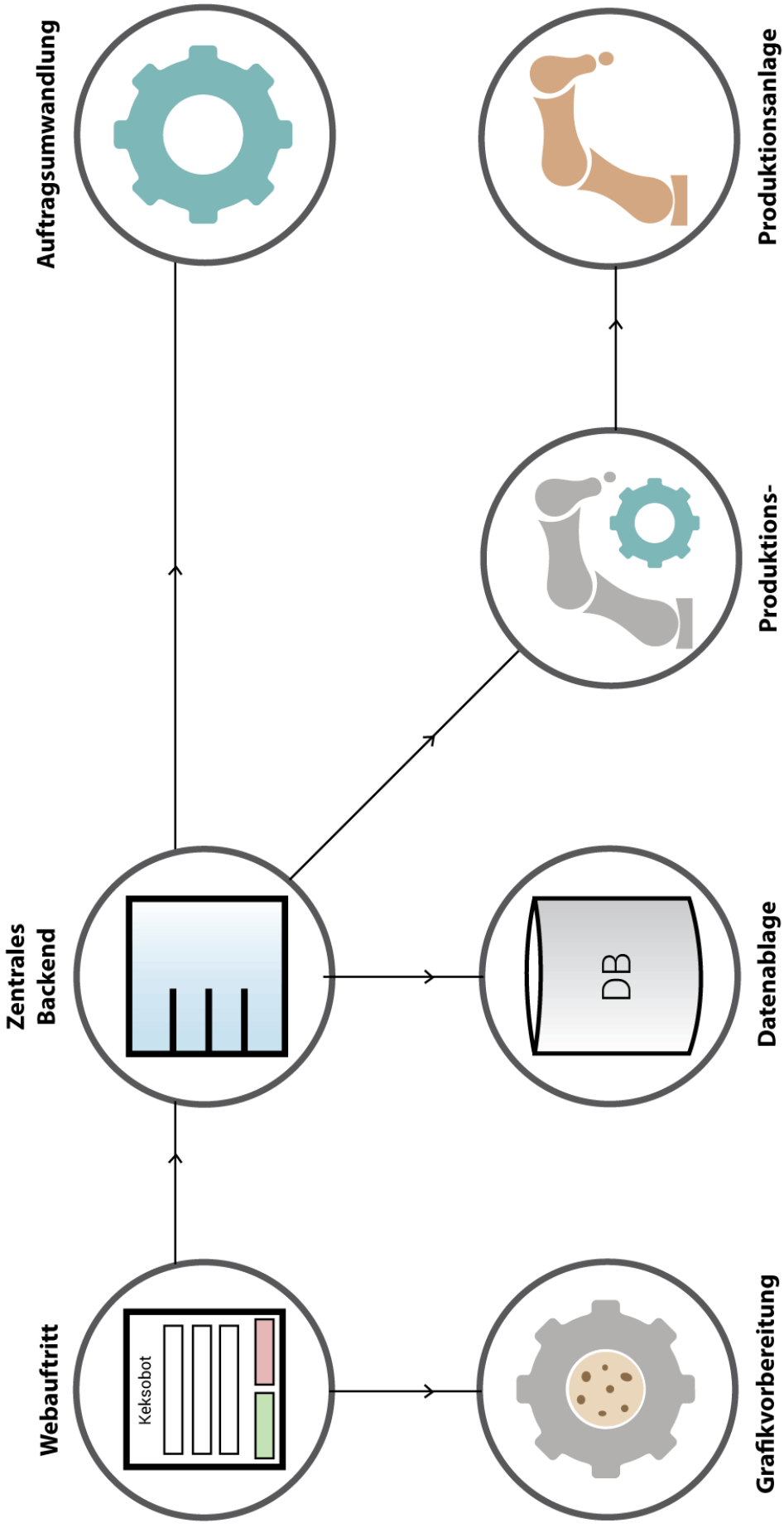


Abbildung 2.1: Die Architektur von Keksobot

Kapitel 3

Konzeptentwicklung

3.1 Serverarchitektur (CM)

Um die Bestellung vom Kunden weiterverarbeiten zu können muss diese zunächst zwischengespeichert werden. Hierfür wird ein Server benötigt. Für seine Architektur gibt es verschiedenste Lösungsvarianten:

3.1.1 Variante 1: Single-Serverhardware

Bei dieser Variante werden alle Dienste auf einem Server betrieben. Das bedeutet, dass es keinen Mehrkostenaufwand gibt, da im Gegensatz zu anderen Varianten, insbesondere zur Verwaltung, weder zusätzliche Hardware noch Software (zum Beispiel: Lizenzen) benötigt wird. Diese Art der Architektur ist hinsichtlich des Datenschutzes riskant, da im Falle eines erfolgreichen Hacker-Angriffs alle Daten ersichtlich sind.

3.1.2 Variante 2: Single-Serverhardware mit Virtualisierung

Hier wird ebenfalls nur ein Server-Gerät benötigt, in dem werden die Dienste in virtuellen Maschinen (VM) betrieben, was das Risiko durch Hacker-Angriffe mindert, da bei einem erfolgreichen Eindringen nur die Daten des sich darauf befindlichen Dienstes offen liegen. Durch die Virtualisierung kommt es zu einem Kostenmehraufwand. Dies kommt unter anderem dadurch zustande, dass eine leistungsstärkere Hardware benötigt wird, da pro Dienst eine virtuelle Instanz des Server erstellt werden muss. Außerdem müssen Lizenzen für die verwendete Virtualisierungssoftware gekauft werden. Aus einer Instanz mit x Diensten wird also eine Instanz die mehrere virtuelle Instanzen, mit je einem Dienst, beinhaltet. Zusätzlich entsteht dadurch ein größerer Wartungsaufwand.

3.1.3 Variante 3: Mehrere Server

Die Verwendung von mehreren Servern (ein Server pro Dienst) ist eine Kombination aus Variante 1 und 2. Hier werden, ähnlich wie in Variante 2, Dienste von einander getrennt. Dies bewirkt, dass das Risiko durch Hacker-Angriffe zwar erhöht wird, da mehr “Angriffsfläche” entsteht, jedoch sind dem Hacker, sollte ein Server erfolgreich gehackt werden, nur ein Teil der Daten ersichtlich. Es entstehen zusätzliche Kosten je nachdem wie viele Server im Endeffekt benötigt werden.

3.1.4 Entscheidung

Im Rahmen des Diplomprojekts ist die Entscheidung auf Variante 1 gefallen, da sie die wenigsten Kosten verursacht. Im Hinblick auf das auftretende Sicherheitsdefizit durch die getroffene Wahl, ist zu sagen, dass

Variante	Single-Serverhardware	Virtualisierung	Mehrere Server
Preis	gering		hoch
Wartungsaufwand	gering	hoch	hoch
Ausreichend Sicher	x	x	x
Ersichtliche Daten(Hacker-Angriff)	alle	teilweise	teilweise

Tabelle 3.1: Übersicht Server Varianten

keine sensiblen Daten[4] auf dem Server persistiert werden müssen. Dazu würden beispielsweise Sexualität, Religion, familiäre Umstände und gesundheitlicher Zustand gehören. Heutzutage werden Kunden bei Zahlung mit den gängigsten Methoden (Kreditkarte, Paypal, Sofortüberweisung) jedenfalls auf gesicherte Seiten der Zahlungsanbieter weitergeleitet, weswegen die Daten hierfür nicht bei der Bestellung übergeben werden müssen.

Im Echtbetrieb wäre allerdings zumindest Variante 2 anzudenken, da zwar keine sensiblen Daten, jedoch personenbezogene Daten, wie zum Beispiel Liefer- und Versandadresse, gespeichert werden.

3.2 Bestellungsannahme (CM)

3.2.1 HTTP-Server

Apache

Apache[14] ist eine der bekanntesten Web-Server Applikationen. Zahlreiche Produkte, wie zum Beispiel: Laravel und Wordpress werden standardmäßig, sprich ohne Verwendung externen Software-Applikationen, unterstützt. Es wird lediglich ein PHP-Interpreter benötigt. Über die Zeit hat sich eine außerordentlich gute Dokumentation entwickelt, welche bei Recherche und zur Problemlösung hilfreich ist. Außerdem unterstützt Apache ein großes Spektrum an Programmiersprachen, welche für die Weiterführung unter Umständen benötigt werden, ohne dafür einen Mehraufwand zu haben.

Nginx

Nginx[33] ist ein jüngeres Produkt, welches ein einfaches Setup und eine simple Final-Konfiguration verspricht. Speziell bei hoher Auslastung wird ein effizientes Ressourcenmanagement gewährleistet. Proxy-Funktionen werden standardmäßig unterstützt, werden jedoch im Rahmen des Diplomprojekts nicht benötigt.

Entscheidung

Die Entscheidung ist auf Apache gefallen, da eine Vielzahl an Sprachen interpretiert werden kann. Zusätzlich können Erweiterungen (Module) effizient und dynamisch geladen werden, was bei richtiger Konfiguration einen Overhead verhindert. Außerdem ist der Apache Web-Server sehr populär, was eine gute Dokumention, Unterstützung von anderen Services und Problemhilfe verspricht. Innerhalb des Teams bestehen bereits fundierte Grundkenntnisse, dadurch fällt die Einarbeitungszeit sehr gering aus.

HTTP-Server	Apache2	NginX
simples Setup	x	x
Erfahrung im Team	x	x
Ausgereifte Dokumentation	x	∼
wenig Overhead	x	x

Tabelle 3.2: Übersicht HTTP-Server

3.2.2 Datenbankmanagementsystem

MySQL

Die kommerzielle Nutzung ist bei MySQL[31] nur mit der Enterprise Edition erlaubt, dies könnte in späterer Folge eine Änderung bedeuten. Abgesehen davon unterstützt MySQL alle notwendigen Funktionen (Backups erstellen, Benutzerrechte individuell verwalten, etc…) für die Webseite. Zusätzlich fällt die Einarbeitungszeit weg, da alle Teammitglieder über grundlegende MySQL-Kenntnisse verfügen.

PostgreSQL

Im Gegensatz zu MySQL benötigt man bei PostgreSQL[36] keine kostenpflichtige Version um es kommerziell zu verwenden. Hier werden ebenfalls alle Funktionen abgedeckt. Konträr zu MySQL wird bei dieser Variante eine Einarbeitungszeit benötigt.

Entscheidung

Da der Datenbankzugriff ohnehin mithilfe eines ORMs[8] stattfindet, stellt der Wechsel zwischen MySQL und PostgreSQL, bei einem Markteintritt, kein großes Problem dar. Für die Entwicklung des Prototyps ist die Lizenzierung allerdings unerheblich. Dies war der ausschlaggebende Punkt warum die Entscheidung auf *MySQL* gefallen ist.

DMBS	MySQL	PostgreSQL
simples Setup	x	x
Erfahrung im Team	x	
Ausgereifte Dokumentation	x	∼
Opensource	x	x
Kommerzielle Nutzung erlaubt	mit zusätzlicher Lizenz	x

Tabelle 3.3: Übersicht Datenbankmanagementsysteme

3.2.3 Backend-Framework

Warum ein Framework?

Durch die Verwendung eines Frameworks werden oft benötigte, komplexe Vorgänge, wie zum Beispiel der abgesicherte Zugriff auf eine Datenbank, enorm vereinfacht. Dies hat zur Folge, dass mehr Arbeit in weniger oder gleicher Zeit erledigt werden kann. Die meisten Frameworks modular aufgebaut, was wiederum einen Overhead verhindert. Daher ist eine einfache Erweiterung möglich, falls essentielle Funktionen nicht "out of the box" untersützt werden. Viele Frameworks verwenden das MVC-Pattern[30], welches zu einem geringeren Wartungsaufwand führt. Darüber hinaus vereinfacht ein Framework meist die Implementierung von Unit-Tests, welche für einen guten Projektausgang unumgänglich sind.

Aufgrund der oben genannten Punkte wurde auch hier ein Framework verwendet. Die Entscheidung ist auf Laravel[24] gefallen, da es alle benötigten Funktionen (gesicherter Zugriff auf die Datenbank, Kommunikation mit externen Services, etc…) abdeckt beziehungsweise durch Module erweitert werden können, und die Gesamt-Einarbeitungszeit geringer ist als bei einem noch nicht bekannten Framework. Zusätzlich ist Laravel sehr gut dokumentiert und hat eine aktive Community, was zur Lösung eventueller Probleme beiträgt.

3.3 Bestellungsverarbeitung (CM)

Um eine möglichste effiziente Abwicklung einer Bestellung zu gewährleisten wurden zwei Varianten evaluiert. Um vorhandene Ressourcen möglichst effektiv einzusetzen wurde großer Wert auf einen effizienten und möglichst ressourcensparenden Ablauf gelegt.

3.3.1 Variante 1: Kettenaufruf einzelner Java-Programme

Nachdem eine Bestellung eingegangen ist, werden übermittelte Daten in der Datenbank persistiert. Anschließend wird ein Java-Programm, welches die vom Kunden angegebene Grafik übernimmt und diese in *roboterneutralen-Code* umwandelt, gestartet. Wobei es irrelevant ist ob es sich um eine Vorlage, welche von uns zur Verfügung gestellt wurde, oder um eine vom Kunden selbst erstellte Grafik, handelt. Dieser wird nicht in der Datenbank gespeichert, da die Wahrscheinlichkeit als gering eingestuft wird, dass genau diese Grafik in Verbindung mit den selben Kekseigenschaften erneut bestellt wird. Zuletzt wird ein weiteres Java-Programm aufgerufen, welches nun den *roboterneutralen-Code* in *roboterspezifischen-Code* umwandelt und diesen an den Roboter übermittelt. Zuletzt beginnt dieser mit der Verzierung der Kekse.

3.3.2 Variante 2: Java-Daemon

Bei dieser Variante werden, genau so wie bei Variante 1, vom Kunden übermittelte Daten, in der Datenbank persistiert. Anschließend wird die Grafik, an einen Java-Daemon übergeben, welcher nun selbstständig alle weiteren Programme aufruft. Hier ist es wieder irrelevant ob es sich um eine Vorlage oder eine vom Kunden angefertigte Grafik handelt. Dies hat den Vorteil, dass nachdem die Grafik übergeben wurde das Backend wieder bereit ist neue Bestellung zu verarbeiten und nicht erst warten muss bis das letzte Java-Programm aufgerufen wurde.

3.3.3 Entscheidung

Die Entscheidung ist auf Variante 2 gefallen, da durch schnellere Entlastung des Php-Backends eine schnelle Abwicklung der eingehenden Bestellung ermöglicht wird. Außerdem kann der Java-Daemon, bei Bedarf, auf einen weiteren Server ausgelagert werden um den Web-Server zu entlasten.

3.4 Keksverzierungen vom Kunden (RG)

außerdemEs ist zu überlegen, in welchem Dateiformat Kunden ihr eigenes Keksdesign an den Verzierungsdienst übermitteln. Die Grundbedingung ist, dass es sich um ein Dateiformat mit Vektorgrafiken handelt. Das kommt daher, da es ist nicht Ziel des Projekts ist, mit dem Roboter z.B. ein Foto auf den Keks übertragen zu können. Stattdessen ist das Ziel, Keksverzierungen anzufertigen, die auch händisch umsetzbar sind.

3.4.1 Bewertungskriterien

Folgende Kriterien sind für die Auswahl des Dateiformats zu betrachten:

- Robotertauglichkeit
- Usability für Kunden
- Verfügbarkeit von Software-Bibliotheken
- Implementierungs-Aufwand

Die Bewertung erfolgt anhand der Abstufung *niedrig*, *mittel*, *hoch*.

3.4.2 Varianten

Für die Gegenüberstellung wurden als Lösungsvarianten ausgewählt:

1. Portable Document Format (PDF)
2. SVG
3. Encapsulated PostScript (EPS)
4. Propritäre Dateiformate (z.B. Adobe Illustrator (AI), AutoCAD native (DWG), CorelDRAW (CDR))

3.4.3 Gegenüberstellung

	PDF	**SVG**	**EPS**	**Propritäre Formate**
Robotertauglichkeit	• Grafikobjekte robotertauglich formuliert • Vektorgrafiken bestehen immer aus Pfaden [16, S. 193] **Daher:** hoch	• vordefinierte Formen sowie Pfade • Auftrennung der Formen für Roboter notwendig **Daher:** mittel	• Grafikobjekte robotertauglich formuliert • Vektorgrafiken bestehen immer aus Pfaden [17, S. 204] **Daher:** hoch	• nicht bekannt, weil keine Dokumentation verfügbar **Daher:** hoch

Usability	• von verbreiteten Grafikeditoren unterstützt • z.B. Adobe Illustrator, Inkscape, GIMP **Daher:** hoch	• von verbreiteten Grafikeditoren unterstützt • z.B. Adobe Illustrator, Inkscape, GIMP • zusätzlich Online-Grafikeditoren, weil Format weborientiert ist **Daher:** sehr hoch	• von verbreiteten Grafikeditoren unterstützt • z.B. Adobe Illustrator, Inkscape, GIMP **Daher:** hoch	• kein gemeinsamer Nenner zwischen Formaten • setzt bestimmtes Softwareprodukt voraus • Verminderung der Menge potenzieller Kunden **Daher:** niedrig
Software-Bibliotheken	• veröffentlichte Format-Definition • weitverbreitet • ausgereifte Bibliotheken verfügbar • z.B. Apache PDFBox **Daher:** hoch	• vergleichbar neues Dateiformat • breite Unterstützung (Web-Browser) • veröffentlichte Format-Definition • daher viel Potenzial für Bibliotheken • z.B. Apache Batik **Daher:** hoch	• Anzahl an Interpretern fällt gering aus • gefundene Bibliotheken: keine Wartung mehr **Daher:** niedrig	• keine offiziellen Interpreter • weil propritär auch keine Drittanbieter **Daher:** niedrig
Implementierungs-Aufwand	• Übersetzung von Figuren zu getrennten Bewegungen nicht mehr notwendig • Filtern von Rastergrafiken • Filtern von nicht-relevanten Funktionen (z.B. Fliesstext, Layouting-Anweisungen, Rastergrafiken) **Daher:** hoch	• Übersetzung von Figuren zu getrennten Bewegungen • Filtern von Rastergrafiken • Filtern von nicht-relevanten Funktionen (z.B. Animationen) **Daher:** mittel	• Übersetzung von Figuren zu getrennten Bewegungen nicht mehr notwendig • Filtern von Rastergrafiken notwendig • Nicht-anwendbare Funktionen (z.B. *Stack, Memory Management*), daher Filteraufwand hoch **Daher:** mittel/hoch	• Interpreter finden bzw. schreiben • Filtern von nicht-relevanten Funktionen (z.B. Rastergrafiken) • sonstige unerwartete Herausforderungen aufgrund mangelnder Dokumentation **Daher:** hoch

Tabelle 3.5: Gegenüberstellung verschiedener Dateiformate für kundeneigene Keksverzierungen

3.4.4 Entscheidung

Es wurde entschieden primär das Vektorgrafik-Format *SVG* zu unterstützen. Die Gründe dafür: gute Lesbarkeit und Nachvollziehbarkeit der Anweisungen, modernes Format mit ausgezeichneter Unterstützung durch Grafikeditoren und Browser, rein als Grafikformat ausgelegt (Vergleich PDF und PostScript: kein gemischter Inhalt).
Durch diese Auswahl wird erwartet eine Unterstützung der kundeneigenen Keksverzierung für den großen Teil der potenziellen Kunden anbieten zu können. Außerdem fällt die Suche nach Software-Bibliotheken zur Verarbeitung von SVG-Dokumenten durch die Popularität leicht. Da es sich um ein vergleichbar

neues Dateiformat handelt, werden diese Bibliotheken oft auch noch aktiv gewartet.

3.4.5 Konzept fuer die Umsetzung

Um die SVG-Dateien von Kunden auslesen zu können, wird die Softwarebibliothek *Apache Batik* verwendet. Die Bibliothek ist in Java geschrieben, was sich mit dem Programmiersprachen-Konzept der Keksobot-Komponenten deckt.

Die Software-Bibliothek übernimmt zur Gänze die Aufgabe des Einlesens, sodass mit dem SVG-Dokument auf objektorientierter Weise interagiert werden kann. *Apache Batik* hilft außerdem insbesondere bei der Verarbeitung von `<path>` Elementen aus SVG, bei denen die Bibliothek die Verarbeitung des sonst aufwendigen *data*-Attributs entwicklerfreundlich umsetzt: die Subbefehle im Pfad werden getrennt ausgegeben, auch dann wenn sie im Dokument in der Kurzform notiert werden. Das hilft bei der Umwandlung zu Roboterbeweungsbefehlen, die getrennt befohlen und ausgefuehrt werden.

Bei SVG muss damit gerechnet werden, dass im Dokument Funktionen verwendet werden, die für die Keksverzierung keine Anwendung haben. Das lässt sich dadurch erklären, dass das Dateiformat sehr von digitalen Ausgabemedien und dem Web geprägt wurde. Auf der anderen Seite ist die SVG-Funktionalität der eingebetteten Rastergrafiken auszufiltern, weil sie aus der Projektdefinition ausgeschlossen wurden. Notwendig ist das Verzichten auf Unterstützung folgender Funktionen:

- Animationen
- Transparenz und Deckkraft
- Filter zur Veränderung der Anzeige (z.B. *Weichzeichner*)
- Farbverläufe als Füllung von Grafikobjekten

Damit Kunden das Ergebnis des Filtervorganges verifizieren können, indem nur die verbliebenen unterstützten Anweisungen seiner SVG-Grafik in einer Vorschau angezeigt werden. So soll eine visuelle Kontrolle durch den Kunden stattfinden.

3.5 Roboteranweisungen in anlagenneutralem Format (MH)

3.5.1 Problemstellung

Die Kommunikation zwischen Preprocessor I und Preprocessor II sollte über ein robotertypneutrales Format ablaufen um dem zukünftigen Kunden die Wahl zu lassen welchen Roboter er verwenden möchte um die Formdaten, gewonnen aus der Benutzereingabe, an den Roboter zu übertragen. Dazu werden folgende Anforderungen an das Format gestellt:

- Möglichst schlanke Dateien (keine Redundanz)
- Spezifizierten Teil um die Metadaten (max Geschwindigkeit, max Druck, usw) zu managen
- Bestenfalls standardisiert
- Einfache Erweiterbarkeit in den 3D Raum

Nach einiger Recherche blieben folgende Optionen über: XML, Step, Gerber.

Übersicht der Lösungsvarianten

Variante 1: Gerber Gerber ist ein standardisiertes Format, welches in der Industrie verwendet wird um Leiterbahnen auf Leiterplatten anzubringen.

Beispiel (Pseudocode)

```
1. Block
| Werkzeug | X-, Y-Koordinate | Steuerfunktionen | Blockende |
2. Block
| Werkzeug | X-, Y-Koordinate | Steuerfunktionen | Blockende |…
usw
```

Es würde sich daher gut zum Abbilden der einzelnen Roboterbewegungen eignen und bietet, durch Layering, die Möglichkeit eine 3. Dimension zu erschaffen.

Variante 2: Step Files Step Files werden im 3D Druck, zur Kommunikation zwischen dem Designprogramm und dem Drucker, verwendet. Sie bilden Produktdaten während den einzelnen Produktionsschritten ab.Es wird also eine Vielzahl verschiedener Step Files benötigt um einen Prozess erfüllen zu können (Zum Beispiel 2 Figuren abbzubilden). Allerdings lässt es sich, sofern alle Anforderungen des jeweiligen ISO Standards erfüllt sind, in eine Vektorgrafik umwandeln.

Variante 3: XML-Files
Bei XML-Files handelt es sich um die flexibelste aller Alternativen, denn sie sind "extensible" (dt.: erweiterbar), lassen sich also auf die jeweiligen Anforderungen anpassen. Ein perfekt auf den jeweiligen Anwendungsfall zugeschnittenes Format ist denkbar.

Gegenueberstellung der Varianten

Nachteile von Gerber als neutrales Format Gegen Gerber spricht die Tatsache, dass einige Dinge beachtet und kommuniziert werden müssen, die kein Äquivalent in der Leiterplattenherstellung haben und daher kein Teil des Gerber Formats sind. Das betrifft zum Beispiel die Steuerung verschiedener I/O's und Geschwindigkeits-, sowie Druck-Informationen die sich bei jedem Materialwechsel ändern können.
Dieses Problem ist auf 2 Arten lösbar : Entweder wird der zur Verfügung stehende Raum (Werkzeug, Steuerfunktion), der im Format enthalten ist, zweckentfremded, was eine sehr hohe Redundanz mit sich bringt. Oder es wird ein extra config-File gefertigt, das zusätzlich ausgelesen werden muss.

Nachteile von STEP als neutrales Format Um mittels Step Informationen abzubilden werden eventuell mehrere Files benötigt. Dieser Umstand ist suboptimal, da wir die gesamte Information in einem Dokument unterbringen müssen. Außerdem ist die STEP-Familie so groß, dass es einen enormen Zeitaufwand benötigen würde um den passenden "Unterstandard" zu finden. Abgesehen davon, unterliegt STEP dem Copyright von ISO und ist *nicht* frei verfügbar.

Nachteile von XML als neutrales Format Aus der Erweiterbarkeit von XML Files erschließt sich, dass sie nicht im engeren Sinne standardisiert sein können. Abgesehen davon ob sie "well-formed" sind lässt sich kein Standard, der nicht zuvor in einem "Schema" festgelegt wurde, prüfen. Dieses zu erstellen verursacht weiteren zeitlichen Aufwand.

Eigenschaft	1 Gerber	2 STEP	3 XML
Kein Overhead		X	X
Standardisierbar	X	X	X
Frei zugänglich	X		X
Erfahrung mit der Handhabung			X
Skalierbar auf große Datenmengen	X	X	X

Tabelle 3.6: My caption

3.5.2 Entscheidung

Insgesamt überwiegen die technischen Vorteile die XML mit sich bringt, trotz des zusätzlichen zeitlichen Aufwandes, weil dem passenden Format, ohne Redundanz aber trotzdem mit allen notwendigen Informationen, eine höhere Gewichtung zugeschrieben wird als der Standardisierung.

3.5.3 Konzept fuer die Umsetzung

Das anlagenneutrale Roboterformat wird von der Auftragsumwandlungs-Komponente erzeugt. Ein XML-Schema wurde für dieses Dokument verfasst. Es wird anschließend erläutert.

```
<?xml version="1.0"?>
<!--  Keksobot XML document schema, Version 2017-Mar-22 -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Order">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ordernr" type="xs:int"/>
        <xs:element name="cookietypeid" type="xs:int"/>
            <xs:element name="amount" type="xs:int"/>

        <xs:element name="Shape" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>

              <xs:element name="metadata">
                <xs:complexType>

                  <xs:sequence>
                    <xs:element name="materialid" type="xs:int"/>

                    <xs:element name="materialReq" minOccurs="0" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:simpleContent>
                           <xs:extension base="xs:string">
                             <xs:attribute name="name" use="required"/>
                           </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>

                </xs:complexType>
              </xs:element>

```

```
            <xs:element name="movement" maxOccurs="unbounded">
              <xs:complexType>

                <xs:sequence>
                  <xs:element name="xstart" type="xs:decimal"/>
                  <xs:element name="ystart" type="xs:decimal"/>
                  <xs:element name="xend" type="xs:decimal"/>
                  <xs:element name="yend" type="xs:decimal"/>
                  <!--Hilfspunkt für Kreisbewegungen, optional außer bei Kreisbewegungen -->
                  <xs:element name="xcurvemiddle" type="xs:decimal" minOccurs="0"/>
                  <xs:element name="ycurvemiddle" type="xs:decimal" minOccurs="0"/>
                </xs:sequence>

                <xs:attribute name="type" use="required">
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:enumeration value="line"/>
                      <xs:enumeration value="circularArc"/>
                      <xs:enumeration value="point"/>
                    </xs:restriction>
                  </xs:simpleType>
                </xs:attribute>


              </xs:complexType>
            </xs:element>
            </xs:sequence> <!-- end of Shape -->

            <xs:attribute name="continuousExecution" use="optional" type="xs:boolean"/>

          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Grundsätzlich wird mit diesem Dokument ein Auftrag (`Order`) beschrieben. Um aus dem XML heraus nachvollziehen zu können, welchen Auftrag das Dokument beschreibt, wird die zugehörige ID aus der Datenbank in das XML-Element `ordernr` kopiert. Natürlich stellt das eine Gefahr der Inkonsistenz dar, wenn sich die ID des Auftrags ändern würde. Allerdings ist die Änderung einer ID in der Praxis nicht zu empfehlen, speziell bei Rückfragen durch Kunden, denen diese ID bekannt ist und die auf ihren Auftrag mit dieser ID referenzieren.

Mit der `cookietypeid` wird definiert für welchen Kekstyp, spezieller aber für welche Keksform, das Anweisungsformat ausgelegt ist. Eine identische Keksverzierung führt zu anderen Roboteranweisungen, wenn ein anderer Keks ausgewählt wurde. Das liegt daran, dass auf zwei Keksen mit anderen Flächeninhalten eben mehr bzw. weniger von der Verzierung verwendet werden kann, sprich die Verzierung durch die Keksgrenzen anders beschnitten wird. Natürlich findet das sogenannte *Clipping* nur statt, wenn die Verzierung überlappt.

Innerhalb eines Auftrags besteht eine Verzierung aus mehreren Formen. Formen dürfen von einander

entfernt auf der Keksfläche liegen, sind also nicht unbedingt miteinander verbunden. Daraus ergibt sich auch der Hinweis für die Auftragsausführung auf der Produktionsebene, dass beim Wechsel der Formen der Verziervorgang gestoppt werden muss.
Jede Form verwendet genau ein Material, das durch seine ID beschrieben wird. Jedes Material legt andere Anforderungen für die Produktionsebene fest, das sind Eigenschaften wie Geschwindigkeit, mit der der Roboterarm bewegt wird, und der Druck, mit dem das Glasurmaterial aufgetragen wird. Diese Anforderungen werden dem Datenbankeintrag für das Material entnommen.

In einer Form finden ein bis viele Bewegungsanweisungen Platz. Jede Bewegungsanweisung hat einen Bewegungstyp und basierend auf diesem Typ werden von der Komponente die dafür notwendigen Eigenschaften angegeben. Ein Kreisbogen zum Beispiel benötigt einen Hilfspunkt um sich eindeutig zu definieren, die anderen Typen allerdings nicht.

Die Bewegungstypen wurden entsprechend dem in der Testumgebung vorliegenden *KUKA KR6 sixx 900 agilus* Roboter ausgewählt, der grundsätzlich Linien, Kreisbögen und Punkt-zu-Punkt-Bewegungen (undefinierte Bahnen) unterstützt. Andere Kurven (z.B. Bezier-Kurven) können durch diesen Robotertyp derzeit nicht eindeutig abgefahren werden.

3.6 Einlesen des anlagenneutralen Formats (MH)

Der zu verarbeitende Input ist das roboterneutrale XML-Format. Es gilt nun, dieses möglichst sinnvoll, also Ressourcen effizient, einfach und richtig, einzulesen, um die einzelnen Informationen (Form, Farbe, Material, Bewegungsart, Menge) weiterverarbeiten zu können.

Variante 1: DOM ("Document Object Model")

Bei DOM handelt es sich um einen W3C-Standard für eine Programmierschnittstelle zur Verarbeitung von XML-Dateien. Hier können XML Dateien sowohl gelesen, als auch geschrieben werden. Kennzeichnend daran ist, dass der Zugriff nicht sequentiell stattfindet. Bei DOM erfolgt er über den sogenannten "DOM-Baum". Daher ist die Arbeit mit DOM intuitiv und komfortabel, eignet sich aber weniger gut für die Verarbeitung extrem großer Dateien, da sich der DOM-Baum an jeder Ebene spaltet und sich die Größe daher schnell vervielfältigen kann.

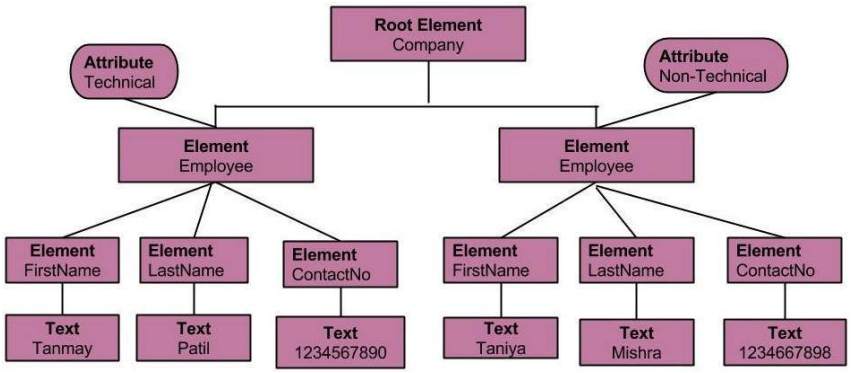


Abbildung 3.1: Beispielhafter DOM-Baum

Eigenschaft	DOM	SAX
Einfach zu verwenden	X	
Lesen möglich	X	X
Schreiben möglich	X	
Erfahrung vorhanden	X	
Auf größere Umfänge skalierbar		X

Tabelle 3.7: Übersicht Einlesen XML

Variante 2: SAX ("Simple API for XML")

Im Gegensatz zu DOM wird bei SAX das XML-Dokuent sequentiell durchlaufen, so können auch sehr große Dokumente, ohne Performanceverlust, bearbeitet werden. Über eine DocumentHandler-Schnittstelle können registierte Callbackfunktionen ereignisgesteuert aufgerufen werden. Es ist in der Verwendung weniger intuitiv als DOM. Hiermit können XML-Dateien ausschließlich gelesen werden.

Gewählte Variante

Wie in der Übersichtstabelle 3.1 ersichtlich, überwiegen die Vorteile bei DOM. Obwohl DOM für größere Dokumente die imperformantere Variante ist, fiel die Entscheidung darauf. Aufgrund der begrenzten Größe der Verzierungen ist die Detailgenauigkeit soweit eingeschränkt, dass es zu keinen Dokumenten, in Größen die die Performance mindern würden, kommen wird. Dieser Umstand wird bereits von den vorangehenden Preprozessoren überprüft. Da DOM nun also keinen Nachteil mitsich bringt ist es durch die intuitivere Verwendung und die vorhandene Erfahrung insgesamt besser als SAX.

Konzept

Um das neutrale Format nun zu verarbeiten wird zunächst das XML-File mittels eines DOM-Parsers eingelesen und in seine Einzelteile zerlegt. Es setzt sich aus *Shapes* und *Movements* zusammen, wobei eine Shape aus einer Menge an Movements besteht. Die Komponenten werden also dementsprechend zusammengefügt und am Ende ein Objekt, das alle Shapes, die ihre jeweiligen Movements enthalten, beinhaltet, an die Produktionsschnittstelle, zurückgegeben. Diese iteriert daraufhin durch das Objekt und teilt dem Roboter die Daten mit, da dieser nicht selbst Files ein-/auslesen kann.

3.7 Roboterkommunikation (MH)

Außerdem

3.7.1 Datenaustauscch zwischen Roboter und Java Programm

Variante 1: JOpenShowVar

[19] JOpenShowVar ist eine Java-Library. Sie kann dazu benutzt werden Variablen am Roboter dynamisch zu setzen. Hier wird ein dynamisches Roboterprogramm also dadurch erreicht, dass dem Roboter bekannte Variablen von außen neu definiert und dann regulär (statisch programmiert) im Roboterprogramm verwendet werden. Dabei stellt die oben erwähnte Softwarebibliothek JOpenShowVar die Verbindung zwischen dem Java-Programm, dass die bereits erwähnten Variablen setzen soll, und dem Roboterprogramm her. Um sie verwenden zu können müssen folgende Funktionen erfüllt sein:

- Vordefinierte Globale Variablen am Roboter

  Diese werden von JOpenShowVar gesetzt

- TCP Port 7000 frei

  Hier läuft die Verbindung zwischen Roboter und Programm

- Socketprogramm

  Um die Verbindung am Roboter herzustellen

Abgesehen vom Setzen der Variablen, lassen sie sich auch auslesen, was einen großen Vorteil für eine automatische Rückmeldung bietet.

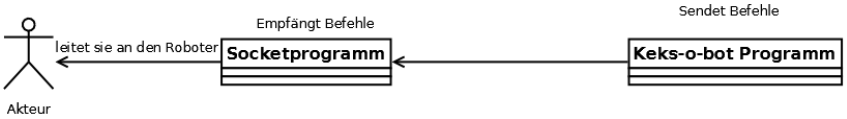


Abbildung 3.2: Zusammenspiel zwischen den Systemen bei JOpenShowVar

Variante 2: Standardisierter Feldbus

Bei dieser Alternative werden die Details für eine Roboteranweisung (Bewegungsart, Geschwindigkeit, Zielposition, …) über einen in der Industrie verwendeten, standardisierten Feldbus ausgetauscht. Um für die gesamte Verzierung alle zugehörigen Anweisungen über den Bus zu übertragen, werden auf die Ausgänge nach jeder fertig ausgeführten Anweisung neue Anweisungsdetails gelegt. Das zugehörige Programm auf der Robotersteuerung pausiert dabei solange das Programm, bis neue Daten eingetroffen sind.

Diese Lösung ist interessant, weil die Komponenten Feldbus-Koppler und Sender der Roboteranweisungen entsprechend den Bedingungen des Kunden angepasst werden können. Es würde nämlich bloß voraussetzen, dass diese beiden Geräte das SPI-Businterface (Serial Peripheral Inteface) implementieren, da es zur Kommunikation zwischen Slave und Master dient. Somit wird die Skalierbarkeit und Modularität des Produktes erweitert, da es sich um ein weit verbreites Interface handelt das sehr viele Roboter unterstützen. Allerdings muss dieser Bus vorhanden sein was zu einem Kostenmehraufwand durch Beschaffung der Hardware und etwaiger Lizenzen führen kann.

Variante 3: KUKA.RobotSensorInterface bzw. KUKA.Ethernet KRL XML

Diese Lösung ist ein Software-Addon vom Hersteller KUKA selbst. Es ermöglicht folgende Funktionen [21, S. 9]:

- Datenaustausch zw. externem System und dem KUKA-Roboter via Ethernet KRL Interface
- Client-, sowie Server-Rolle ist definierbar (ob KUKA-Roboter oder externes System)
- Transfer von binären Daten oder XML Strings

Dabei wird ein *externes System* von der Produktbeschreibung eher als Signal oder Messergebnis eines Sensors definiert. Daher ist die Verwendung einer Software an der Stelle des externen Systems (quasi eine Simulation “Sensor”), die dann über den gemeinsamen Kommunikationskanal “Sensordaten” (bei uns Bewegungsdaten) sendet, eigentlich ein Workaround.

Eigenschaft	1 JOpenShowVar	2 Feldbus	3 Kuka Interface
Einfach	X		X
Lesen	X	X	X
Schreiben	X	X	X
Erfahrung mit der Handhabung	X		
Skalierbar auf große Datenmengen	X	X	X
Kostenlos	X		
Kompatibel mit allen Robotern		X	

Tabelle 3.8: Übersicht Roboterverbindungsmethoden

Positiv ist an dieser Lösung, dass sie von KUKA entwickelt wird. Das ist von Vorteil, weil das in der Entwicklungsphase beispielhaft entwickelte, spezialisierte Endprodukt "verzierender Roboter" mit einem KUKA-Roboter (Steuerung *KR C4 compact*) arbeiten wird. Außerdem ist zu erwarten, dass die Lösung gute Stabilität bietet.

Negativ ist aber, dass die Lösung schon vom Konzept her KUKA-spezifisch ist. Die Logik ist in der Industrie nicht standardisiert (im Gegensatz zu Eingängen/Ausgängen von Feldbus-Systemen). Deshalb muss bei einem Wechsel der Robotersteuerung ein Umstieg auf eine komplett andere Logik vorgenommen werden, nicht nur der Hardware-spezifische Kommunikationsteil in der keks-o-bot ausgetauscht werden.

KUKA.Ethernet KRL XML ist ein kostenpflichtiges Add-on.

Unter den Umständen, dass sehr ähnliche Funktionlität auch von den anderen Varianten oder einem Feldbus-System erreicht werden kann, wird diese Variante nicht empfohlen. Auch, weil die Recherche nicht leicht fiel und die Machbarkeit mit dieser Variante deshalb nicht klar ist.

Gewählte Variante

Aufgrund der oben genannten Punkte, fiel die Wahl auf die JOpenShowVar-Library. Sie bietet alle geforderten Features, abgesehen von der sofortigen Übertragbarkeit auf andere Robotersysteme, und eignet sich daher gut für die zukünftigen Zwecke.

Konzept

Zur Umsetzung werden verschiedene Variablen am Roboter definiert. Unter anderem wird die Art der Bewegung , die Geschwindigkeit, das Material und die Start und End Punkte aller Movements, sowie etwaige Metadaten (Ist der Roboter aufnahmefähig, fährt er gerade, o.ä.) benötigt. Diese Daten werden von der Java-Software automatisch an den Roboter übertragen indem die definierten Variablen am Roboter immer wieder beschrieben werden.

Kapitel 4

Zentrales Backend (CM)

4.1 Sicherheit

4.1.1 Datenbank-Sicherheit

Aus Sicherheitsgründen kann auf die Datenbank nur von *localhost* zugegriffen werden. Das bedeutet, dass nur der Server, auf dem sie tatsächlich gehostet ist, Zugriff hat.

Um das zu bewirken wurden die Berechtigungen des *root-Users* so modifiziert, dass er nur von der IP-Adresse des Servers aus auf die Datenbank zugreifen kann:[2]

```
/*Entfernen aller Zugriffsrechte*/
REVOKE ALL ON *.* FROM 'root'@'%';

/*Zuweisen der neuen Zugriffsrechte*/
GRANT ALL PRIVILEGES ON *.* TO root @'localhost' IDENTIFIED BY 'mysecret';
GRANT ALL PRIVILEGES ON *.* TO root @'127.0.0.1' IDENTIFIED BY 'mysecret';
```

Listing 4.1: root-User Rechte einschränken

Außerdem wird der MySQL-DIenst auf die IP-Adresse *127.0.0.1* gebunden, welche den Server (*localhost*) repräsentiert.[2]

```
#Konfigurationsdatei: /etc/mysql/my.cnf
[...]
[mysqld]
bind-address = 127.0.0.1
[...]
```

Listing 4.2: Binden des MySQL-Servers auf localhost

Mit der Zeit hat es sich durchgesetzt, dass es für jeden Service, der auf die Datenbank zugreift, einen eigenen User gibt. Das geschieht wiederum aus Sicherheitsgründen, da der User des zugehörigen Services ausschließlich Zugriff auf Daten hat, welche er auch benötigt.[6]

```
CREATE USER 'laravel'@'localhost' IDENTIFIED BY 'mypass';
GRANT ALL PRIVILEGES ON webshop.* TO laravel @'localhost' IDENTIFIED BY 'mysecret';
```

Listing 4.3: Einschränken der Benutzer Rechte

In diesem Beispiel wird der User *laravel* erstellt und anschließend die Rechte so gesetzt, dass er nur auf die Datenbank *webshop* Zugriff hat.

4.1.2 HTTPS

Hyper Text Transfer Protocol Secure[15] (HTTPS) wird verwendet um Daten verschlüsselt vom Client an den Server zu übertragen. Dadurch wird gewährleistet, dass selbst wenn ein Angreifer die Verbindung abhorcht, er die Daten nicht interpretieren kann, da sie verschlüsselt sind.

Um den aktuellen Sicherheitsstandards gerecht zu werden, wurde ein HTTPS-Zertifikat[25] von `https://letsencrypt.org/` angefordert und mit den Projektdomains (`https://keksobot.online/` und `https://keksobot.tech/`) verbunden.

4.1.3 Serversicherheit

Jeder offene Port bei einem Serversystem[26] ist ein Risiko, deshalb wurden alle nicht benötigten Ports blockiert um die Angriffsfläche für mögliche Hacker so gering wie möglich zu halten.

```
#installieren der Firewall
sudo apt-get install ufw

#Zeigt den Status der Firewall an
sudo ufw status

#erlaubt den Zugriff via SSH (Port 22), welcher benötigt wird um den Server zu warten
sudo ufw allow ssh

#gibt den Port 80 frei, sodass der Webserver  von außerhalb erreicht werden kann
sudo ufw allow http

#Der Port 443 wird benötigt um eine HTTPS Verbindung zu ermöglichen.
sudo ufw allow 443

#schaltet die Firewall "scharf". Alle eben konfigurierten Regeln werden nun eingehalten
sudo ufw enable
```

Listing 4.4: Firewall Konfiguration

Um einer weiteren Sicherheitslücke[32] vorzubeugen wurde das Login mit einem Passwort deaktiviert[13]. Das bedeutet, dass ein Verbindungsaufbau nur via SSH-Keys[39] möglich ist.

```
#Öffnen des Files in einem Editor
vim /etc/ssh/sshd_config

#Diese optionen müssen in dem eben geöffnenten File gesetzt werden
ChallengeResponseAuthentication no
PasswordAuthentication no
UsePAM no

#Um die Einstellung neu zuladen, muss der SSH-Service neugestartet werden
/etc/init.d/sshd restart
```

Listing 4.5: Passwort Login deaktivieren

4.2 Einführung in Laravel

4.2.1 Database-Model

Ein Model beziehungsweise ein Objekt von diesem Model repräsentiert eine Tabelle in der Datenbank. Mithilfe diesem können DB-Anfragen getätigt werden, was den gesamten Zugriff sehr vereinfacht. Anstatt selbstgeschriebene SQL-Anfragen zu schreiben wird lediglich ein Objekt instanziert und die zur Verfügung gestellten Methoden aufgerufen.

```
//Instanzieren des Models
$cookie_query = new Cookie;

//Anfrage an die Datenbank
$cookieBase = $cookie_query
    //Sucht einen "runden" Keks mit dem Namen "Butterkeks"
    ->where('cookie_name', 'Butterkeks')
    ->where('cookie_shapecategory', 'rund')
    //Liefert das erste Ergebnis zurück
    ->first();
```

Listing 4.6: Einfacher Datenbank Zugriff via DB-Model

Dieser beispielhafte Zugriff auf eine Datenbanktabelle liefert den ersten Keks zurück bei dem der *cookie_name* gleich “Butterkeks” und die *cookie_shapecategory* gleich “rund” ist.

Models können in Laravel mithilfe von einem Automaten erstellt werden. Dieser Automat erzeugt eine Datei welche von der Klasse Model erbt und somit Zugriff auf alle vorhandenen Methoden hat.

```
php artisan make:model Cookie
```

Listing 4.7: Erstellen eines Models in Laravel

Um das Model nun später verwenden zu können, müssen in der vom Automaten erstellten Datei, welche sich im *app/* Ordner befindet, einige Attribute gesetzt werden:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Cookie extends Model
{
    /**
    [...]
    */

    protected $table = 'cookie_bases';
    $primaryKey = ['cookie_name', 'cookie_shapeCategory'];
    $incrementing = false;
    protected $fillable = [
    'cookie_name', 'cookie_shapeCategory', 'cookie_svg', 'allergens', 'price', 'weight', 'height',
    ];
}
```

Listing 4.8: Beispiel DB-Model

- $table - beinhaltet den Namen der Tabelle in der die Daten des Objektes persistiert werden

- $primaryKey - beinhaltet alle Primärschlüssel-Felder der Tabelle (Ein Primärschlüssel ist ein beziehungsweise mehrere Werte mit welchen der Datensatz eindeutig identifiziert werden kann.)
- $incrementing - Ein künstlicher Primärschlüssel (zum Beispiel eine ID) ist standardmäßig autoincrementing, was bedeutet, dass er sich bei jedem neuen Eintrag in der Tabelle automatisch um eins erhöht. Dieser Automatismus kann durch das setzen der Variable auf *false* deaktiviert werden.
- $fillable - Um dem Model mitzuteilen welche Felder die Tabelle besitzt müssen diese hier eingetragen werden.

4.2.2 Migrations und Seeds

Migrations werden in der Praxis unter anderem dazu verwendet um DB-Tabellen zu erstellen und zu modifizieren. Sie werden auch für eine Art Versionskontrolle verwendet. Das bedeutet, dass es möglich ist, nachdem die Tabelle via Migrations verändert wurde, diese auf den vorherigen Stand zurückzusetzen. Laravel bietet auch für das Erstellen von Migration-Dateien einen Automaten an.

```
php artisan make:migration CreateUsersTable
```

Listing 4.9: Erstellen einer Migration

In dieser eben erstellten Datei, welche sich im *database/migrations* befindet, kann nun ein Blueprint (Entwurf beziehungsweise Bauplan) der Tabelle erstellt werden.

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration
{
    /**
    [...]
    */
    public function up() {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('firstname');
            $table->string('lastname');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
    [...]
    */
    public function down() {
        Schema::drop('users');
    }
}
```

Listing 4.10: Erstellung eines Blueprint via Migrations

Diese Beispiel Migration erstellt eine Tabelle mit dem Namen *users*. In dieser sind folgende Felder vorhanden:

- id (Primärschlüssel)
- firstname
- lastname
- email (dieser Wert darf ähnlich wie ein Primärschlüssel nur einmal vorkommen)

$table ist ein Objekt des Blueprints welches einige Methoden besitzt um Felder mit bestimmten Datentypen und Funktionen zu erstellen. Welche zur Verfügung stehen kann in der offiziellen Dokumentation[23] von Laravel nachgelesen werden.

Um die erstellten Migrations in die Datenbank zu migrieren muss folgender Befehl ausgeführt werden:

```
#nur beim ersten Mal
php artisan migrate:install
php artisan migrate
```

Listing 4.11: Migrieren einer Tabelle

Zwei weitere oft verwendete Befehle sind:

```
#mit Hilfe dieses Befehls kann der vorherige Stand wiederhergestellt werden.
php artisan migrate:rollback

#löscht alle Tabellen in der Datenbank.
php artisan migrate:reset
```

Listing 4.12: Weitere Migration Befehle

Seeds werden dazu verwendet beispielhafte Daten, zum Beispiel für Testzwecke, in die Tabelle einzufügen. Laravel bietet auch hier wieder einen Automaten an.

```
    php artisan make:seed UserSeed
```

Listing 4.13: Erstellen eines Seeds

In den eben erstellen Seed, welcher sich im Ordner *database/seeds/* befindet, müssen nun die Beispieldaten eingetragen werden. Hier ein Beispiel Seed:

```
<?php

use Illuminate\Database\Seeder;

class UserSeed extends Seeder
{
    /**
    [...]
    */
    public function run()
    {
        DB::table('users')->insert([
        'firstname' => 'Christoph',
        'lastname' => 'Miko',
        'email' => 'christoph.miko.96@gmail.com',
        'password' => bcrypt('mysecret'),

```

```
        ]);
    }
}
```

Listing 4.14: Beispielseed

Der erstellte Seed muss nun im SeedController eingetragen werden damit er beim nächsten Seedvorgang berücksichtigt wird. Der SeedController befindet sich im Ordner *database/seeds/* und heißt *DatabaseSeeder.php*

```
$this->call(UserSeed::class);
```

Listing 4.15: Eintragen des Seeds in den SeedController

Ähnlich wie bei den Migrations wird der Seed erst mittels Konsolenbefehl in die DB-Tabelle eingefügt.

```
    php artisan db:seed

    #Wird verwendet um gleichzeitig zu migrieren und zu seeden
    php artisan migrate --seed
```

Listing 4.16: Einfügen eines Seeds in die Datenbank

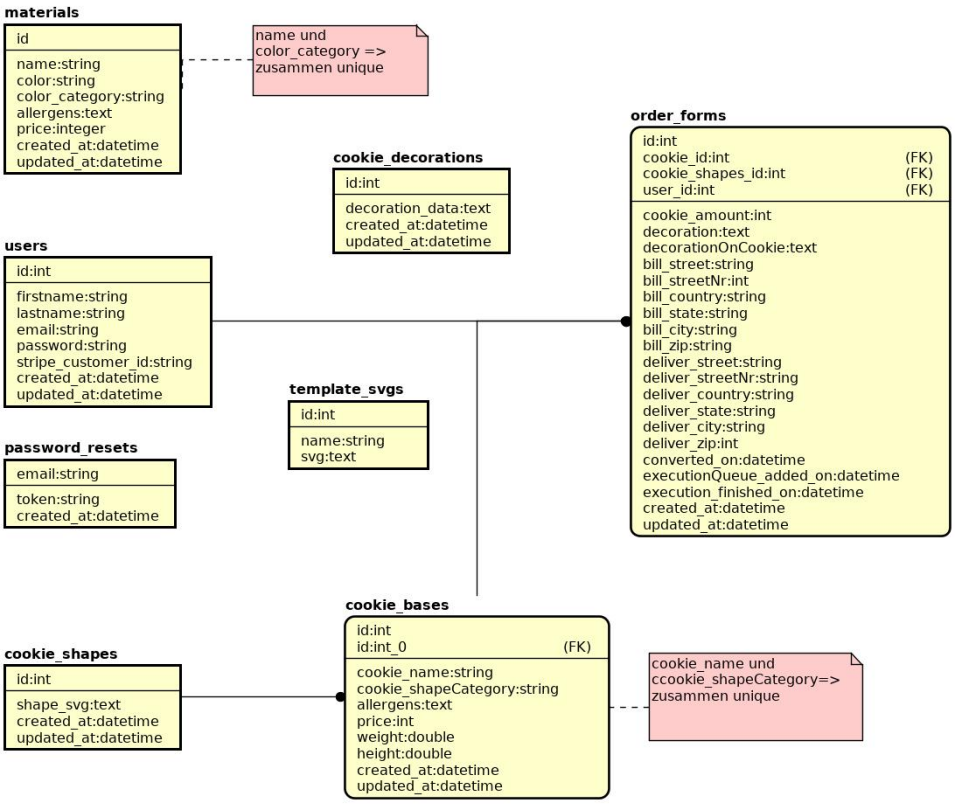


Abbildung 4.1: ER Diagramm des Datenmodels von Keks-o-bot

Projektspezifische Seeds und Migrations befinden sich im Anhang (Kapitel 12) der Diplomarbeit.

4.2.3 Views

Views sind Dateien welche jenen Code beinhalten der zu einem späteren Zeitpunkt als Website angezeigt werden soll. In welcher Form dieser Code geschrieben ist, ist hierbei egal, er muss lediglich von einem Browser interpretierbar sein.

Laravel bietet ein Templatingsystem names *Blade* an, welches verwendet wird um PHP-Code in einer einfacheren Form zu schreiben.

```
//Blade-Code
Hello, {{ $name }}.

//PHP-Code
echo 'Hello, ' . $name;
```

Listing 4.17: Blade vs PHP

Die Variable $name kann entweder eine vorher erstellte oder eine vom Controller übergebene sein.

Außerdem unterstützt Blade Vererbung was die Verwendung von *Layouts* ermöglicht, um anschließend *Sections* darin auszugeben.

```
@extends('layouts.cookieshop') // Gibt an, welches Layout die View erweitert

@section('content') //Zwischen den Sections-Tags steht der Kontent der zu einem späteren
    Zeitpunkt ausgegeben wird
```

```
5         Hallo
      @endsection
7
      @yield('content') // Wenn die View nun von einem Controller aufgerufen wird, wird das Layout +
          der Kontent  angezeigt
```

Listing 4.18: Layout und Sections

Blade bietet einige Funktionen welche in der offiziellen Dokumentation[22] beschrieben sind.

4.2.4 Routen

Laravel achtet auf Veränderungen der URI (Uniform Resource Identifier) im Zusammenhang mit einem bestimmen HTTP-Verb[47].

- GET - Es wird eine Ressource (zum Beispiel eine Datei) vom Server angefordert
- POST - Schickt Daten zur Weiterverarbeitung an den Server
- PUT - Wird verwendet um eine Datei auf den Server, unter Verwendung der URI, hochzuladen.
- DELETE - Die angegebene Ressource wird gelöscht
- PATCH - Ermöglicht es ein Dokument, ohne es wie bei PUT vollständig zu löschen, zu aktualisieren
- OPTIONS - Gibt eine Liste mit Methoden und Merkmalen des Servers zurück

Routen werden in der *web.php*-Datei, welche sich im Ordner *routes/* befindet, definiert.

```
Route::get('/', function () {
2     return view('welcome');
});
4
Route::group(['middleware' => 'web'], function () {
6     Route::get('/login',array('as' => 'login', 'uses' => 'RenderController@showLogin'));
});
```

Listing 4.19: Beispiel Route

Im obigen Beispiel (siehe Listing 4.19) wird eine Route erstellt welche auf einen GET-Request horcht. Sobald die angegebene URI in Kombination mit dem richtigen HTTP-Verb aufgerufen wird, führt sie die in ihr implementierte Logik aus. In diesem Fall wäre das die *View* mit dem Namen *welcome* anzuzeigen. Es wird auch von einer anonymen Route gesprochen, da sie keinen Namen besitzt.

Im zweiten Beispiel wird der Route ein Name (*login*) zugewiesen und sie ruft einen Controller (*RenderController*) auf. Der Aufruf wird zusammen mit dem Namen in einem Array der Route übergeben und die Methode *showLogin* aufgerufen.

Routen können mittels der *group-Methode* in Gruppen eingeteilt werden. Eine Gruppe kann verschiedene Eigenschaften haben wie zum Beispiel die Zuweisung einer *Middleware*. In diesem exemplarischen Codeabschnitt wird der Route die *Middleware web* zugewiesen was bedeutet, dass sie Zugriff auf *Cookies, CSRF-Token (Cross-Site-Request-Forgery)* und die *PHP-Session* hat.

- Cookies - Ein Cookie wird durch den Server (Website) auf dem Client (Betrachter) platziert und beinhaltet beliebige Informationen.
- CSRF-Token Der Token wird bei jeder Anfrage mitgeschickt und anschließend auf seine Richtigkeit überprüft. Dies bietet Schutz vor CSRF [50], was eine Attacke ist bei der der Client vom Angreifer

gezwungen wird Handlungen bei dem Dienst bei dem er aktuell authentifiziert ist, durchzuführen.

- PHP-Session - In der PHP-Session können Informationen jeglicher Art gespeichert werden. Sie wird gestartet sobald eine Website aufgerufen wird und häufig mit Beenden des Browsers geschlossen.

4.2.5 Controller

Im Controller befindet sich die Hauptlogik der Webapplikation (in nativen PHP geschrieben). Er wird von Routen aufgerufen. Um einen Controller zu erstellen kann der von Laravel zur Verfügung gestellt Automat verwendet werden. Alle Laravel Controller befinden sich im Ordner *app/Http/Controllers*.

```
1 php artisan make:controller Name
```

Listing 4.20: Erstellen eines Controllers

Controller werden neben dem Ausführen "der Hauptlogik" zum Beispiel zum Entgegennehmen und Verarbeiten von übermittelten Daten aus einem Formular auch verwendet, um *Views* anzuzeigen und Informationen an diese weiter zu geben.

```
1 $errormsg = "Es ist ein Fehler aufgetreten";
  return view('home')->with(['error_message' => $errormsg);
```

Listing 4.21: Anzeigen einer View mittels Controller

In diesem Beispiel wird eine *View* mit dem Namen *home* angezeigt, welche eine Fehlermeldungen mit dem Wert "Es ist ein Fehler aufgetreten" übergeben bekommt. Diese Fehlermeldung kann dann im Frontend mittels *Blade* ausgegeben werden.

Der Zugriff auf die PHP-Session funktioniert mit einer von Laravel zur Verfügung gestellen Helperklasse. Sie ermöglicht Werte sehr simpel hinzuzufügen oder zu modifizieren (bearbeiten, löschen, usw…)

```
    // Es wird eine Variable (beispiel_Wert)  mit dem Wert Wert1 in der Session angelegt.
2   $request->session()->put('beispiel_Wert', 'Wert1');

4   //Die Variable beispiel_Wert wird mit einem neuen Wert ('bearbeiteter-Wert1') versehen.
    $request->session()->put('beispiel_Wert', 'bearbeiteter-Wert1');
6
    //Die Variable beispiel_Wert wird aus der Session ausgelesen.
8
    $tmp_var = Session('beispiel_Wert')
10
    //Löscht eine Variable aus der Session
12  $request->session()->forget('beispiel_Wert');
```

Listing 4.22: Zugriff auf die PHP-Session

Die, durch den Controller angezeigten, *Views* hat keinen direkten Zugriff auf die Session. Um allerdings Informationen dennoch Anzeigen zu können kann der *View*, wie oben beschrieben, eine Variable mitgegeben werden, welche in späterer folge ausgegeben und angezeigt wird.

4.2.6 Login/Registrierung automatisch erstellen

Da Login und Registrierung sehr häufig verwendet werden, stellt Laravel einen Automaten zur Verfügung welcher ein Basis-Login-System erzeugt. Dies beinhaltet alle benötigten Routen (Login, Logout,

Registrieren) und die dazu gehörigen Controller. (Login-, Register-, Logout, ForgotPassword-, ResetPasswordController)

```
php artisan make:auth
```

Listing 4.23: Erstellen des Basis-Login-Systems

4.2.7 Installation des Stripe-Plugins für Laravel

Stripe[41] ist ein Unternehmen, welches sich auf Zahlungsabwicklungen spezialisiert hat. Es agiert im Falle Keks-o-bot als Mittelsmann für Kreditkarten Zahlungen, sorgt für eine sichere Verbindung und abstrahiert die Zahlungs-Daten des Benutzers. Es wird lediglich eine von Stripe generierte Abfolge von Zeichen (*Token*) zurückgeliefert, welche die Kreditkarte repräsentiert. Mit diesem *Token* kann diese in weiterer Folge **einmalig** belastet werden.

Zuerst muss ein Konto bei Stripe angelegt werden. Im Rahmen der Diplomarbeit wurde es so konfiguriert, dass nicht mit echtem Geld gearbeiten werden kann, allerdings kann das Konto jederzeit auf ein Geschäftskonto erweitert werden.

Um Stripe-PHP via composer zu installieren muss folgender Befehl ausgeführt werden:

```
composer require stripe/stripe-php
composer update
```

Listing 4.24: installation von Stripe-PHP via Composer

Nachdem Stripe-PHP erfolgreich installiert wurde, muss der *Test-Secret- Key* und der *Test Publishable Key*, welcher aus dem Stripe Benutzerprofil entnommen werden kann, in die Environment-Datei von Laravel eingetragen werden.

```
STRIPE_KEY=pk_test_xxxxxxxxxxxxxxxxxxxxx
STRIPE_SECRET=sk_test_xxxxxxxxxxxxxxxxxx
```

Listing 4.25: Eintragen der StripeKeys in der Laravel Environment-Datei

Diese Keys dürfen unter keinen Umständen geteilt oder veröffentlicht werden!!

Zuletzt muss der Treiber von Stripe im Framework aktiviert werden. Das geschiet in der *services.php*-Datei welche sich im Ordner *config/* befindet

```
'stripe' => [
    'model' => App\User::class,
    'key' => env('STRIPE_KEY'),
    'secret' => env('STRIPE_SECRET'),
],
```

Listing 4.26: Eintragen des Stripe-Treibers in Laravel

4.3 Infopage

4.3.1 Login

Um eine Kommunikation zwischen Front- und Backend zu gewährleisten wurde definiert wie der HTTP-Request aussehen muss. Er beschreibt in welcher Form die Daten an das Backend geschickt werden

müssen, um sie richtig interpretieren zu können

```
_token=JLo2e3ziOWr96bo0lqEDewHFYXzLxfBfFloCMmiy
email=christoph.miko.96%40gmail.com
password=mysecret
remember=on
```

Listing 4.27: HTTP-Request (Login)

- _token - CSRF-Token welcher bei jedem Formular mitgeschickt werden muss
- email - E-Mail Adresse des Benutzers, welche er bei der Registrierung angegeben hat
- password Passwort des Benutzers, welches er bei der Registrierung angegeben hat
- remember - Gibt an ob ein Cookie gesetzt werden soll, damit der User bei einem erneuten Besuch der Seite angemeldet bleibt oder nicht

Die Daten werden an folgende Route übermittelt:

```
Route::post('login', ['as' => 'login.post', 'uses' => 'Auth\LoginController@login']);
```

Listing 4.28: POST-Route Login

Nachdem die Daten erfolgreich an das Backend übermittelt wurden, wird validiert ob es sich um einen vollständigen HTTP-Request handelt.

```
/**
* Validate the user login request.
*
* @param  \Illuminate\Http\Request  $request
* @return void
*/
protected function validateLogin(Request $request) {
    $this->validate($request, [
        $this->username() => 'required', 'password' => 'required',
    ]);
}
```

Listing 4.29: Validierung Login-Request

Im Falle einer fehlerhaften Validierung wird der Benutzer automatisch auf die vorherige Seite umgeleitet und eine entsprechende Fehlermeldungen angezeigt.

Im günstigen Fall einer erfolgreichen Validierung wird überprüft ob die Daten zu einem Benutzer in der Datenbank zugeordnet werden können.

```
    /**
* Attempt to log the user into the application.
*
* @param  \Illuminate\Http\Request  $request
* @return bool
*/
protected function attemptLogin(Request $request) {
    return $this->guard()->attempt(
        $this->credentials($request), $request->has('remember')
    );
}

/**
```

```
    * Get the needed authorization credentials from the request.
15  *
    * @param  \Illuminate\Http\Request  $request
17  * @return array
    */
19  protected function credentials(Request $request) {
        return $request->only($this->username(), 'password');
21  }
```

Listing 4.30: Überprüfen der Benutzerdaten

"Guard" ist die von Laravel integrierte "Wache", welche für den Anmeldeprozess essentiell ist. Sie überprüft die Anmeldedaten, mit Hilfe der Datenbank. War diese Überprüfung erfolgreich, so wird der Benutzer zum Webshop weitergeleitet und kann dort sein Benutzerkonto verwalten oder eine Bestellungen tätigen.

Im Falle einer fehlgeschlagenen Überprüfung wird der Benutzer ebenfalls auf die vorherige Seite umgeleitet.

Laravel unterstützt Login *throttling*. Daher wird der Benutzer nach einer bestimmtem Anzahl an fehlerhaften Anmeldungen, für eine bestimmte Zeit, aus dem System ausgeschlossen, da davon auszugehen ist, dass er böse Absichten hat.

```
1   /**
    * Handle a login request to the application.
3   *
    * @param  \Illuminate\Http\Request  $request
5   * @return \Illuminate\Http\Response
    */
7   public function login(Request $request) {
        $this->validateLogin($request);
9
        // If the class is using the ThrottlesLogins trait, we can automatically throttle
11      // the login attempts for this application. We'll key this by the username and
        // the IP address of the client making these requests into this application.
13      if ($this->hasTooManyLoginAttempts($request)) {
            $this->fireLockoutEvent($request);
15
            return $this->sendLockoutResponse($request);
17      }

19      [...]

21      // If the login attempt was unsuccessful we will increment the number of attempts
        // to login and redirect the user back to the login form. Of course, when this
23      // user surpasses their maximum number of attempts they will get locked out.
        $this->incrementLoginAttempts($request);
25
        return $this->sendFailedLoginResponse($request);
27  }
```

Listing 4.31: Throttelt-Login Mechanismus

4.3.2 Registrierung

Hier wurde, ähnlich wie beim Login (Abschnitt Unterabschnitt 4.3.1), der Aufbau des HTTP-Requests definiert.

```
firstname=Christoph
lastname=Miko
email=christoph.miko.96@gmail.com
password=mysecret
password_confirmation=mysecret
```

Listing 4.32: HTTP-Request (Registrierung)

Die Daten werden an folgende Route übermittelt:

```
Route::post('register', ['as' => 'register.post', 'uses' => 'Auth\RegisterController@register']);
```

Listing 4.33: POST-Route Registrierung

Nach der erfolgreichen Übermittlung der Daten an das Backend werden diese ebenfalls validiert. Ob die Daten valide oder invalide sind sagt wiederum nichts über die Richtigkeit, sondern nur über die Gültigkeit, der Daten aus.

```
/**
* Get a validator for an incoming registration request.
*
* @param  array  $data
* @return \Illuminate\Contracts\Validation\Validator
*/
protected function validator(array $data) {
    return Validator::make($data, [
        'firstname' => 'required|max:255',
        'lastname' => 'required|max:255',
        'email' => 'required|email|max:255|unique:users',
        'password' => 'required|min:6|confirmed',
    ]);
}
```

Listing 4.34: Validierung Register-Request

- firstname - Vorname des Benutzers (Maximal 255 Zeichen lang und ist verpflichtend)
- lastname - Nachname des Benutzers (Maximal 255 Zeichen lang und ist verpflichtend)
- email - E-Mail Adresse des Benutzers (Maximal 255 Zeichen, muss eine valide E-Mail Adresse sein, ist verpflichtend und darf nur einmal in der DB-Tabelle "Users" vorkommen)
- password - Passwort des Benutzers (Minimum 6 Zeichen, ist verpflichtend und muss mit dem *password_confirmation* übereinstimmen)
- password_confirmation - erneute Eingabe des Passworts um einen Fehler auszuschließen (ebenfalls verpflichtend)

Falls die Validierung fehlschlägt wird der Benutzer auf die vorherige Seite umgeleitet und eine, an den Fehler angepasste, Fehlermeldungen angezeigt.

Anderenfalls wird der Benutzer angelegt und zum Webshop weitergeleitet.

```
/**
* Create a new user instance after a valid registration.
```

```
*
* @param  array  $data
* @return User
*/
protected function create(array $data) {
    return User::create([
        'firstname' => $data['firstname'],
        'lastname' => $data['lastname'],
        'email' => $data['email'],
        'password' => bcrypt($data['password']),
    ]);
}
```

Listing 4.35: Anlegen eines Benutzers

Das Passwort wird zuvor mit dem *bcrypt-Algorithmus* verschlüsselt und erst dann in die Datenbank persistiert.

4.3.3 Accountverwaltung

Nachdem der Benutzer sich angemeldet hat, kann er seine Benutzerdaten nachträglich über die Accountverwaltung ändern. Hier gelten die gleichen Richtlinien wie bei der Registrierung (Unterabschnitt 4.3.2). (Zum Beispiel darf die E-Mail Adresse nur einmal verwendet werden, da diese genutzt wird um den Benutzer eindeutig zu identifizieren. Der HTTP-Request ist ebenfalls der gleiche)

Die Daten werden an folgende Route übermittelt:

```
Route::post('/shop/updateProfile', ['as' => 'post_updateProfile', 'uses' => '
    UserProfileController@updateProfile']);
```

Listing 4.36: POST-Route Accountmanagement

```
$this->validate($request, [
    'firstname'         =>      'string',
    'lastname'          =>      'string',
    'email'             =>      'email|unique:users',
    'password'          =>      'min:6|confirmed'
]);
```

Listing 4.37: Validierung Accountmanagement

Bei einem fehlgeschlagenen Änderungsversuch wird der Benutzer wieder auf die Accountmanagement-Seite umgeleitet und eine, an den Fehler angepasste, Fehlermeldungen angezeigt

Nach erfolgreicher Validierung werden die Benutzerdaten in der Datenbank geändert, allerdings nur wenn die übertragenen Werte (die neuen Werte) nicht leer sind. Nachdem die neuen Werte in der Datenbank persistiert wurden, wird der Benutzer wieder zu der Accountmanagement-Seite umgeleitet und eine Erfolgsmeldung erscheint.

```
$user = Auth::user();
[...]
//Überprüfen ob der übermittelte Wert nicht leer ist
if($request->input('email') != '') {
    $user->email = htmlentities($request->input('email'), ENT_COMPAT, 'UTF-8');
}
```

```
8  //Überprüfen ob der übermittelte Wert nicht leer ist
   if ($request->input('firstname') != '') {
10     $user->firstname = htmlentities($request->input('firstname'), ENT_COMPAT, 'UTF-8');
   }
12
   //Überprüfen ob der übermittelte Wert nicht leer ist
14 if ($request->input('lastname') != '') {
       $user->lastname = htmlentities($request->input('lastname'), ENT_COMPAT, 'UTF-8');
16 }

18 //Überprüfen ob der übermittelte Wert nicht leer ist
   if ($request->input('password') != '') {
20     $user->password = bcrypt($request->input('password'));
   }
22 //Speichern der veränderter Werte
   $user->save();
24
   //umleiten des Benutzers auf die Accountmanagement-Seite
26 return redirect()->route('get_updateProfile')->with('update_success', "Profile successfully updated"
       );
```

Listing 4.38: Persistieren der neuen Werte in der Datenbank (Accountmanagement)

- Die statische Methode *Auth::user()* liefert den akutell eingeloggten Benutzer zurück.
- mit $user->xy wird dem konkreten User-Attribute ein neuer Wert zugewiesen
- htmlentities werden verwendet damit keine speziellen “Tags” in der Datenbank gespeichert werden können. Zum Beispiel wird ein “<” durch “<” ersetzt. Das bietet einen umfangreichen Schutz vor Cross-site scripting, auch kurz XSS genannt. Dabei handelt es sich um einen Angriff bei der ein Client side-script in zum Beispiel eine Website injiziert wird. Dieses Script wird in weiterer Folge von anderen Besuchern der Website automatisch und ungewollt ausgeführt falls er auf den gleichen Datenbank Eintrag zugreift.

Die statische Methode *Auth::user()* liefert den akutell eingeloggten Benutzer zurück.

4.4 Webshop

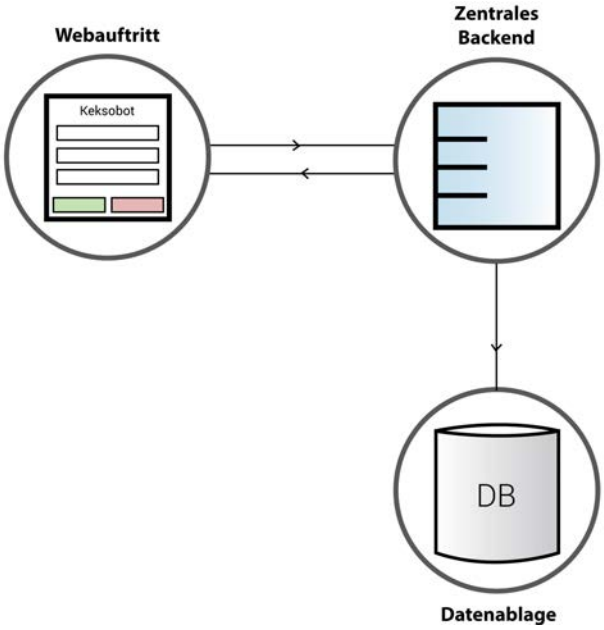


Abbildung 4.2: Datenfluss zwischen Datenablage Webauftritt, zentralen Backend und Datenablage

Nach jedem Schritt den der Kunde im Bestellformular abschickt, erfolgt ein Datenaustausch zwischen *Webauftritt* und dem *zentralem Backend*. Es werden Daten, welche der Kunde übermittelt hat, validiert, für eine Weiterverarbeitung aufbereitet und anschließend entsprechende Meldungen zurückgeschickt. Zuletzt wird der Auftrag in der *Datenablage* persistiert.

4.4.1 Schritt 1 - Auswählen der Grafik

Um die verfügbaren Keksarten (zum Beispiel: Butterkeks, Mürbteigkeks, etc…) und Keksformen (zum Beispiel: Blume, rund, etc…) im Frontend anzeigen zu können, müssen diese aus der Datenbank ausgelesen und an das Frontend übergeben werden.

Verwendeter HTTP-Request Aufbau und Route:

```
cookie_name=name des Kekses
cookie_shapecategory=Form des
cookie_amount=Anzahl
customer_choice=Template / eigene Verzierung
user_svg=Falls eigene Verzierung hochgeladen wurde wird dieses hier übertragen

//Route
Route::post('/shop/order/stepOne', array('as' => 'post_stepOne', 'uses' => '
    OrderProcessController@postStepOne'));
```

Listing 4.39: HTTP-Request und Route Schritt 1

Nachdem der Benutzer eine Keksart, Keksform und eine Grafik (kundeneigene, oder Template) ausgewählt, abgeschickt und diese validiert wurden, werden diese vom Backend entgegen genommen und zwischengespeichert.

Falls die Validierung fehlschlägt wird der Benutzer auf die vorherige Seite umgeleitet und eine, an den Fehler angepasste, Fehlermeldungen angezeigt.

Es gilt als “bad practice” wenn ein und der selbe Code mehrfach im Programm vorhanden ist. Um das zu vermeiden werden einige Algorithmen in Methoden ausgelagert.

Da bei fast jedem Schritt eine Validierung statt findet, wurde auch das automatische Weiterleiten in eine Fehlermeldung ausgelagert.

```
static function redirectUserBackWithErrorMessage ($msg) {
return Redirect::back()->with('email', Auth::user()->email)->withErrors([$msg])->withInput();
}
```

Listing 4.40: Umleitung mit Fehler

Eine neue SVG-Grafik wird erstellt, welche sich aus dem SVG des Basiskekses und der Verzierungsgrafik zusammensetzt.

```
/* construct request XML using chosen cookie shape and user decoration */
$preprocessingRequest = new \DOMDocument;
$preprocRootEle = $preprocessingRequest->createElement("DecorationPreprocessingRequest");
$preprocessingRequest->appendChild($preprocRootEle);

// set element for cookie shape outline
$preprocCookieshape = $preprocessingRequest->createElement('cookieOutline');
$preprocCookieshapeData = $preprocessingRequest->createCDATASection($cookieOutline);
$preprocCookieshape->appendChild($preprocCookieshapeData);
$preprocRootEle->appendChild($preprocCookieshape);

// set element for decoration (template was received from DB query)
$template_decoration = Template_svg::find($chosen_template)['attributes']['svg'];
$preprocUserDecoration = $preprocessingRequest->createElement('decorationData');
$preprocUserDecorationData = $preprocessingRequest->createCDATASection($template_decoration);
$preprocUserDecoration->appendChild($preprocUserDecorationData);
$preprocRootEle->appendChild($preprocUserDecoration);

// set 'preprocessingFeature' element because Templates only need rescaling
$preprocFeatureRescale = $preprocessingRequest->createElement('preprocessingFeature');
$preprocFeatureRescale->textContent = "RESCALE";
$preprocRootEle->appendChild($preprocFeatureRescale);

/* transform XML DOM tree to single-line String for socket delivery */
$preprocessorRequestString = $preprocessingRequest->saveXML();

// remove line-breaks because preprocessor expects one line only
$preprocessorRequestString = preg_replace("/\\r\\n?|\\n/", '', $preprocessorRequestString);

/* build socket connection to decoration preprocessor, send request */
$kbotProps = KeksobotPropertiesFile::getPropertiesArray();
```

Listing 4.41: Zusammensetzten des SVG-Files

Anschließend wird ein externes Java-Programm, via Sockets, aufgerufen und diesem die eben erstellte SVG-Grafik übergeben.

```
$preprocesserSocket = fsockopen($kbotProps['KBOT_DECORATION_PREPROCESSOR_HOST'], $kbotProps['
    KBOT_DECORATION_PREPROCESSOR_PORT']);
fwrite($preprocesserSocket, $preprocessorRequestString . "\r\n");
fflush($preprocesserSocket);
$preprocessing_result = fgets($preprocesserSocket);
fclose($preprocesserSocket);
```

Listing 4.42: "Übermitteln der SVG-Datei via Sockets

Im Falle einer kundeneigenen Grafik wird der volle Umfang des Grafikverarbeitung-Programms ausgeführt. Wurde allerdings ein Template ausgewählt so wird nur die *Rescaling-Komponente* aufgerufen.

Zuletzt werden alle verarbeiteten Werte (Keksart, Keksform, vorverarbeitetes SVG-Dateien) in der PHP-Session abgelegt um darauf später Zugriff zu haben und der Benutzer zum nächsten Schritt weitergeleitet.

4.4.2 Schritt 2 - Auswählen des Verzierungsmaterials

Diese Methode überprüft ob der Benutzer berechtigt ist den Schritt, welchen er versucht aufzurufen, ausführen darf. Das wird ab Schritt 2 jedes mal überprüft, deshalb wurde sie ebenfalls ausgelagert.

```
static function checkIfCurrentUserStep ($s) {
return (session('current_Order.currentStep') == $s or session('current_Order.currentStep') == '
    doneOnce');
}
```

Listing 4.43: Überprüfen ob der Benutzer berechtigt ist diesen Schritt durchzuführen

Um die SVG, welche im Schritt 1 (Unterabschnitt 4.4.1) bearbeitet wurde, im Frontend anzeigen zu können, muss die Höhe und Breite aus der SVG ermittelt und anschließend dem Frontend übergeben werden. Weiters müssen die Verzierungsmaterialen aus der Datenbank ausgelesen und ebenfalls an das Frontend übergeben werden.

```
$svg_xmlObject = simplexml_load_string($svg);
$svg_xmlDom = new \DOMDocument;
$svg_xmlDom->loadXML($svg);
$parent = $svg_xmlDom->documentElement;
$svg_viewbox = $parent->getAttribute("viewBox");
$svg_viewbox_data = explode(' ', $svg_viewbox); // [0] = x, [1] = y, [2] = width, [3] = height

$material = new Material;
$all_materials = $material->select('id','name', 'color_category', 'allergens')->get();
$all_materials_array = $all_materials->toArray();
```

Listing 4.44: Auslesen der viewbox und Verziehrungsmaterialien

Verwendeter HTTP-Request Aufbau und Route:

```
figurs_material_choice=Liste mit den Ausgewählten Materialien (index des Arrays entpricht der
    Teilverzierungs ID)

//Route
Route::post('/shop/order/stepTwo', array('as' => 'post_stepTwo', 'uses' => '
    OrderProcessController@postStepTwo'));
```

Listing 4.45: HTTP-Request und Route Schritt 2

Nach dem der Benutzer die zu verwendenden Verzierungsmaterialien ausgewählt und abgeschickt hat, wird eine Liste angefertigt welche die Teilform mit dem jeweiligen Verzierungsmaterial beinhaltet. Diese wird zuletzt in die PHP-Session geschrieben und der Benutzer wird zu Schritt 3 weitergeleitet

```
$materialChoice = $request->figurs_material_choice;
$materialChoice = json_decode($materialChoice);

/*find material id to name and color_category*/
$material = new Material;
$figure_material_map = array();
```

```
foreach($materialChoice as $choice) {
    $tmp = $material->where('color_category', htmlentities($choice[2]))->where('name', htmlentities(
        $choice[1]))->first();
    $figure_material_map[$choice[0]] = $tmp->id;
}
```

Listing 4.46: Erstellen der Form-Material-Liste

4.4.3 Schritt 3 - Positionieren der Verzierung auf dem Keks

Die SVG-Grafik wird erneut mit konkreter Höhe und Breite angezeigt.

Verwendeter HTTP-Request Aufbau und Route:

```
repositionedDeco=SVG-File als String mit den neuen Positionswerten

//Route
Route::post('/shop/order/stepThree', array('as' => 'post_stepThree', 'uses' => '
    OrderProcessController@postStepThree'));
```

Listing 4.47: HTTP-Request und Route Schritt 3

Nachdem der Benutzer die Verzierung auf dem Keks positioniert und abgeschickt hat wird der zugehörige Wert für die SVG-Grafik in der PHP-Session aktualisiert. Eine Weiterleitung auf Schritt 4 erfolgt.

```
/* save repositioned deco (clean svg & without cookie) into Session var */
$request->session()->put('current_Order.decoration', $cleanDecoSVGString);
$request->session()->put('current_Order.decorationOnCookie', $repositionedDecoOnCookie);
```

Listing 4.48: SVG-Wert in PHP-Session aktualisieren

4.4.4 Schritt 4 - Erfassung der Rechnungs- und Lieferadresse

Zunächst wird überprüft ob der Benutzer schon etwas bezüglich Liefer- oder Rechnungsadresse zu einem früherer Zeitpunkt bekannt gegeben hat (zum Beispiel im Falle einer Korrektur). Sollte dies der Fall sein werden dem Frontend die bereits zuvor angegeben Daten übermittelt.

```
if(Session::has('current_Order.deliver_billing_info')) {
    return view('sections.cookieshop.orderProcess.stepFour', ['email' => Auth::user()->email, '
        deliver_billing_info' => session('current_Order.deliver_billing_info')]);
} else {
    return view('sections.cookieshop.orderProcess.stepFour', ['email' => Auth::user()->email]);
}
```

Listing 4.49: Überprüfen auf bereits zuvor angegebene Versanddaten

Falls die Rechnungsadresse identisch mit der Lieferadresse ist muss nur die Rechnungsadresse angegeben werden. Im anderen Falle sind beide Angaben verpflichtend.

```
$this->validate($request, [
    'bill_street'   =>      'required|string',
    'bill_streetNr' =>      'required|integer',
    'bill_country'  =>      'required|string',
    'bill_state'    =>      'required|string',
    'bill_city'     =>      'required|string',
```

```
    'bill_zip'      =>      'required|integer',
]);
if($request->input('deliver_is_bill') == 'on') {
    $this->validate($request, [
        'bill_street'   =>      'string',
        'bill_streetNr' =>      'integer',
        'bill_country'  =>      'string',
        'bill_state'    =>      'string',
        'bill_city'     =>      'string',
        'bill_zip'      =>      'integer',
    ]);

} else {
    $this->validate($request, [
        'deliver_street'   =>      'required|string',
        'deliver_streetNr' =>      'required|integer',
        'deliver_country'  =>      'required|string',
        'deliver_state'    =>      'required|string',
        'deliver_city'     =>      'required|string',
        'deliver_zip'      =>      'required|integer',
    ]);
}
```

Listing 4.50: Validierungsmaske der Versanddaten

Verwendeter HTTP-Request Aufbau und Route:

```
//Rechnungsadresse:
bill_street=Straße
bill_streetNr=Straßen Nummer
bill_country=Land
bill_state=Staat
bill_city=Stadt
bill_zip=Postleitzahl

deliver_is_bill=Ob die Rechnungs- und Lieferadresse gleich sind

//Lieferadresse
deliver_street=Straße
deliver_streetNr=Straßen Nummer
deliver_country=Land
deliver_state=Staat
deliver_city=Stadt
deliver_zip=Postleitzahl

//Route
Route::post('shop/order/stepFour', array('as' => 'post_stepFour', 'uses' => '
    OrderProcessController@postStepFour'));
```

Listing 4.51: HTTP-Request und Route Schritt 4

Nachdem der Benutzer seine Versanddaten eingetragen, beziehungsweise korrigiert und abgeschickt hat.Wrden sie, falls sie valide sind, in der PHP-Session gespeichert. Er wird zu Schritt 5 weitergeleitet.

4.4.5 Schritt 5 - Übersicht und Bezahlung

Nun werden alle gesammelten Daten, plus der berechnete Preis an das Frontend übergeben und dort angezeigt. Zusätzlich wird die Möglichkeit geboten die Versanddaten zu korrigieren falls diese fehlerhaft angegeben wurden. Zu diesem Zweck wird wieder zu Schritt 4 umgeleitet.

Nachdem der Benutzer die von ihm getätigten Eingaben überprüft hat, wird ein Checkout mit Kreditkarte, von Stripe[40] verwaltet, angeboten.

Stripe agiert hier als Mittelsmann. Es übernimmt die vom Benutzer eingegeben Zahlungsdaten, überprüft diese auf ihre Gültigkeit und liefert, im Falle einer erfolgreichen Inspektion, einen *stripe-token* und die vom Benutzer verwendete *stripe-email*, welche er im Zuge des Bezahlvorgangs angegeben hat, an das Backend zurück. Mit Hilfe dieses *stripe-tokens* kann die verbunde Kreditkarte belastet werden.

Damit Stripe die übermittelten Daten richtig interpretieren kann, muss der von Stripe vorgegebene Aufbau des HTTP-Requests eingehalten werden.

```
Stripe::setApiKey(config('services.stripe.secret'));

$customer = Customer::create([
    'email' => request('stripeEmail'),
    'source' => request('stripeToken'),

]);

$charge = Charge::create([
    'amount' => $price,
    'customer' => $customer->id,
    'currency' => 'eur'
]);
```

Listing 4.52: Kreditkarten Belastung via Stripe

Nach erfolgreicher Bezahlung werden alle vom Benutzer gesammelten Daten in der Datenbank persistiert.

Zuletzt wird er auf eine Seite weitergeleitet welche eine Erfolgsmeldung anzeigt.

Kapitel 5

Vorverarbeitung von Verzierungsgrafiken (RG)

5.1 Einführung

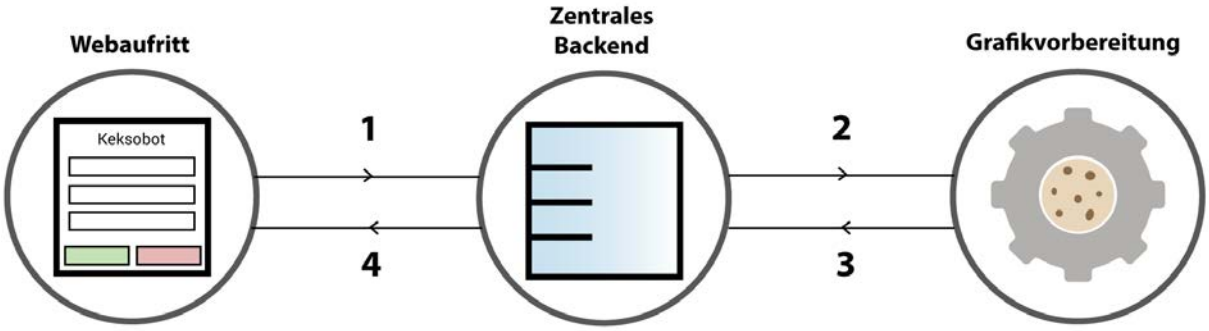


Abbildung 5.1: Datenfluss zwischen Webshop, HTTP-Backend und Vorarbeitungs-Komponente

1. Der Kunde lädt eine Verzierungsgrafik hoch oder wählt eine vorhandene aus
2. Das HTTP-Backend leitet die Verzierungsgrafik an die Grafikverarbeitungs-Komponente
3. Eine an die Keksobot-Umgebung angepasste Version der Verzierungsgrafik wird zurückgegeben ODER im im Fehlerfall die Fehlermeldung
4. Das HTTP-Backend leitet das Ergebnis der Vorverarbeitung an die Benutzerschnittstelle; die Keksverzierung wird zur Verifikation durch den Kunden angezeigt bzw. im Fehlerfall wird dieser dem Kunden berichtet

Alle Verzierungsgrafiken im Dateiformat SVG müssen einige Grafikverarbeitungsschritte durchlaufen, bevor sie für die Keksverzierung verwendet werden können. Der Grund ist, dass vom Kunden stammende Verzierungsgrafiken erst geprüft und bei Bedarf an die Keksobot-Arbeitsumgebung angepasst werden müssen. Der selbe Verarbeitungsschritt ist aber auch bei Katalogverzierungen des Dienstleisters notwendig, da der Keksobot-Administrator nicht notwendigerweise über die technischen Voraussetzungen Bescheid weiß.

Bei kundeneigenen Verzierungen ist es möglich, dass die übertragene Grafik fehlerhaft ist — das muss erkannt und an den Kunden berichtet werden, damit dieser eine Korrektur vornehmen kann.
Auch möglich ist es, dass die Grafik zu gross ist oder nicht korrekt im Koordinatensystem liegt. Einige Keksobot-Komponenten setzen bestimmte Eigenschaften der Keksverzierung voraus, damit diese ihre Aufgabe korrekt ausführen. Der Webshop beispielsweise setzt ein korrekt positionierte sowie skalierte

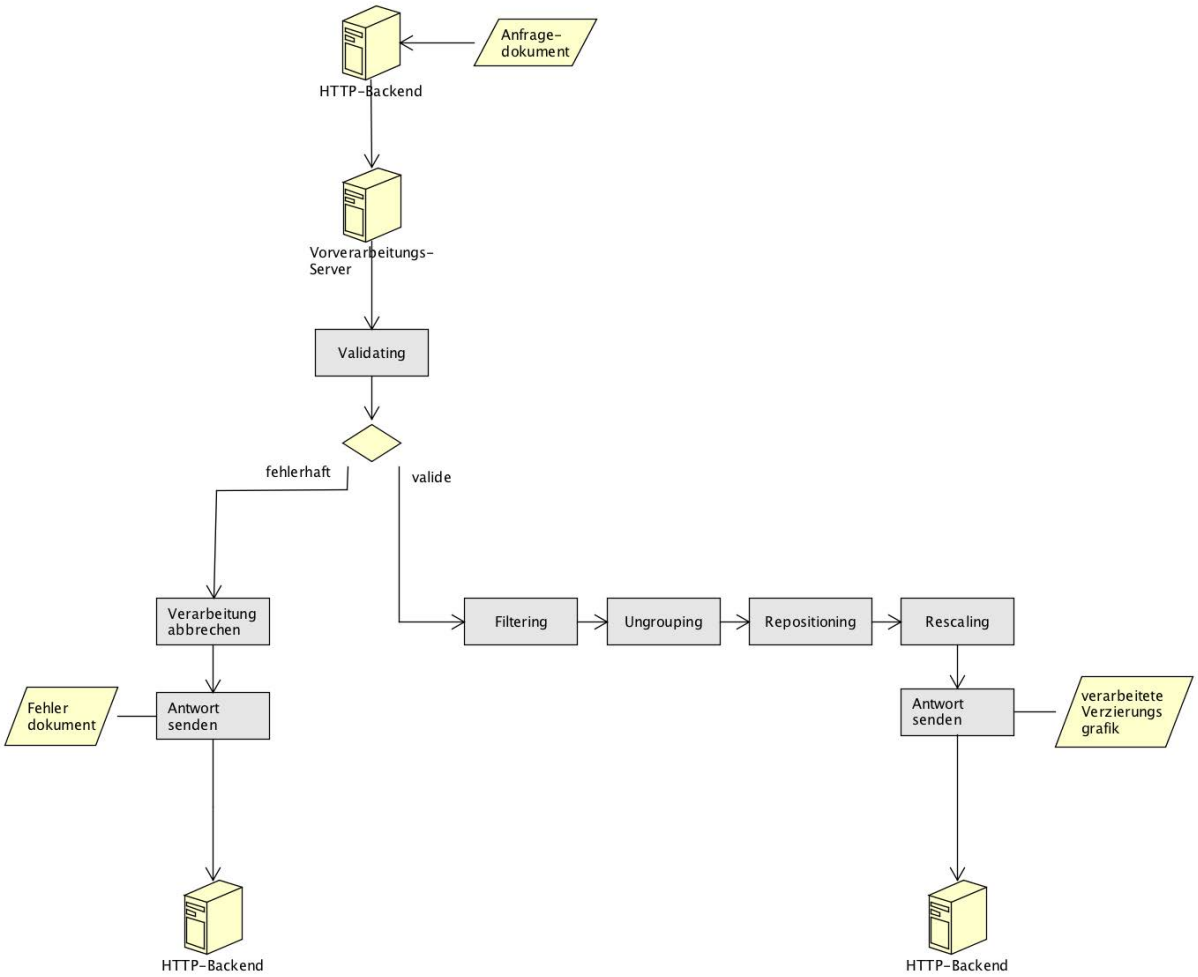


Abbildung 5.2: Prozess der vollständigen Keksverzierungs-Vorverarbeitung

Grafik voraus, um eine korrekte Vorschau zu erreichen.

Bei Katalogverzierungen des Dienstleisters wird beim Einlesen einer neuen Verzierung der Satz an Grafikverarbeitungs-Schritten angewendet. Allerdings ist der letzte Schritt, das Skalieren der Verzierung, erst nach der Auswahl der Keksform möglich. Nachdem der Kunde die Auswahl getroffen hat, wird die Grafikverarbeitungs-Komponente für das Neuskalieren aufgerufen.

5.2 Server-Daemon

Die Grafikverarbeitungs-Komponente wird als “Daemon” umgesetzt, weil mehrere Kundenanfragen zur gleichen Zeit eintreffen können. Ein “Daemon” ist ein Programm, das an einer definierten IP-Adresse und einem definierten Port neue Aufträge annimmt und jede Kundenanfrage in einem eigenen Thread bearbeitet. Ein Nachteil des Daemon-Ansatzes ist, dass bei einem gröberen Programmfehler in einem einzelnen Thread, möglicherweise die komplette Verarbeitungs-Komponente ausfällt. Durch ausführliche Behandlung von typischen Fehlerfällen konnte diese Wahrscheinlichkeit reduziert werden, ist aber bei einem zuvor unbekannten Fehler nicht auszuschliessen. Wenn das Keksobot-Softwarepaket auf die empfohlene Weise installiert wurde, wird der Ausfall der Grafikverarbeitungs-Komponente erkannt und die Software automatisch neu gestartet.

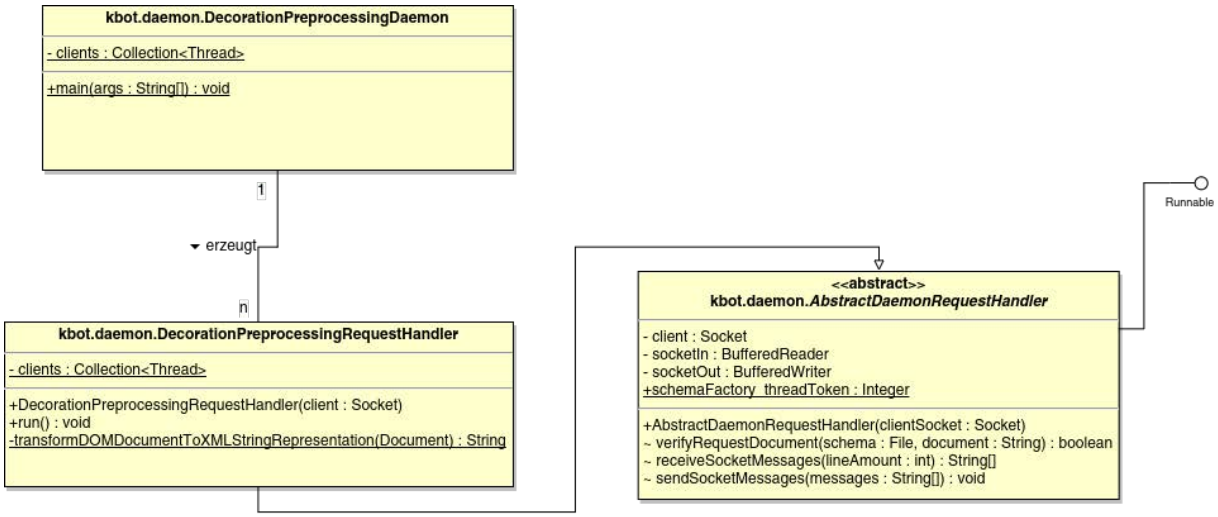


Abbildung 5.3: Klassenstruktur der Auftragsannahme

Im Vergleich zum Daemon-Ansatz würde bei getrennten Instanzen der Grafikverarbeitungs-Komponente ein Ausfall der Software nur einen Kundenauftrag betreffen. Diese Variante bringt aber andere Nachteile, etwa hoher Ressourcenbedarf durch mehrere Instanzen der Java Virtual Machine (JVM), mit sich.

Der Port, auf dem neue Aufträge angenommen werden, ist in der Datei `Keksobot.properties` konfigurierbar. Gleichzeitig wird diese Datei verwendet, um die HTTP-Backendkomponente für den Aufruf der Grafikverarbeitungs-Komponente zu konfigurieren.

```
1 KBOT_DECORATION_PREPROCESSOR_HOST=127.0.0.1
  KBOT_DECORATION_PREPROCESSOR_PORT=60210
```

Listing 5.1: Auszug aus Keksobot.properties zur Server-Socket-Konfiguration

Aus Abbildung 5.3 geht hervor, dass die *Daemon*-Klasse die "Mutterklasse" darstellt und eine Ansammlung an Threads der Klasse `DecorationPreprocessingRequestHandler` verwaltet. Pro eingelangtem Request wird eine Instanz dieser *Handler*-Klasse eröffnet.

5.2.1 Ablauf der Vorverarbeitung

Innerhalb der Klasse `DecorationPreprocessingRequestHandler` wird zunächst das Request-Dokument aus dem Socket ausgelesen. Es wird vorausgesetzt, dass das komplette Dokument in einer Zeile gesendet wird (die mit einem `LF`-Zeichen endet). Um einen Übertragungsfehler oder einen Programmfehler im HTTP-Backend zu erkennen, wird das Dokument nun gegen das zugehörige Schema verglichen. Ist das Dokument valide, werden die ausgewählten Verarbeitungs-Schritte angewendet. Ist es fehlerhaft, wird ein Fehlerdokument als Antwort zurückgesendet. In Abbildung 5.2 wird der Ablauf grafisch dargestellt.

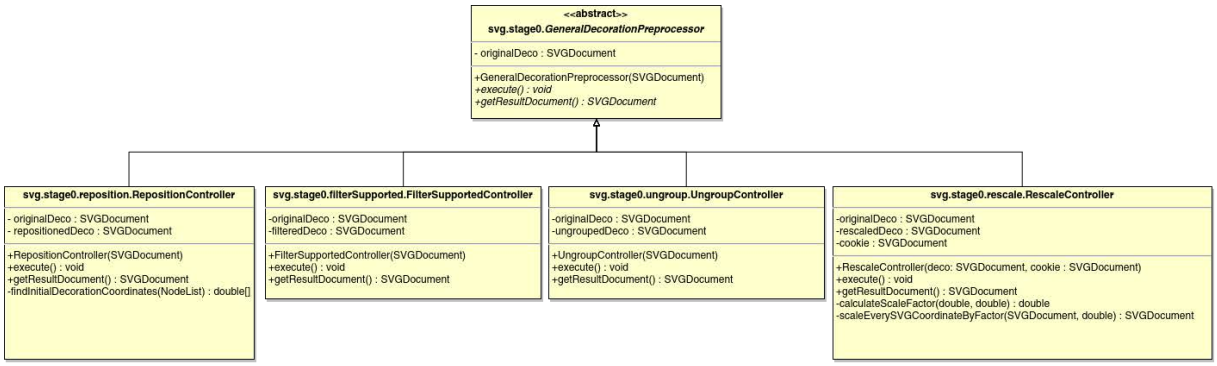


Abbildung 5.4: Preprocessing-Module in der Übersicht

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="DecorationPreprocessingRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cookieOutline" type="xs:string"/>
                <xs:element name="decorationData" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

Listing 5.2: Das Anfragedokument der Grafikverarbeitung

Jeder Verarbeitungsschritt bzw. jedes Preprocessing-Modul erbt von der abstrakten Klasse `svg.stage0.GeneralDecorationP` und erhält dadurch ein einheitliches Interface. Der Request-Handler startet von außen die Verarbeitung durch das Grafikverarbeitungs-Modul und ruft das Ergebnis ab. Dieses gibt er an das nächste Modul weiter. Das entspricht der Verkettung von Klassen mit gleicher Schnittstelle, wie es beim *Decorator*-Softwarepattern der Fall ist.

Die verarbeitete Verzierung wird anschliessend wieder in ein Antwortdokument verpackt und einzeilig an den Auftraggeber (also das HTTP-Backend) übertragen. Ist ein Fehler aufgetreten, dann wird keine Verzierungsgrafik mitgegeben. Der Statuscode deutet auf einen Verarbeitungsfehler hin.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="DecorationPreprocessingResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="decorationData" type="xs:string" minOccurs="0"/>
            </xs:sequence>

      <xs:attribute name="status" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="OK"/>
            <xs:enumeration value="INPUT_INVALID"/>
            <xs:enumeration value="PREPROCESSING_ERROR"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Listing 5.3: Das Antwortdokument der Grafikverarbeitung

5.2.2 Starten der Komponente

Die Grafikverarbeitungs-Komponente wird, wie die anderen Keksobot-Komponenten auch, über Servicedateien des Init-Systems *systemd* gestartet und verwaltet. Der Name der Servicedatei lautet nach einem typischen Installationsvorgang `kbot-decorationPreprocessor.service`.
Nach der ersten Inbetriebnahme der Keksobot-Komponenten sollten die Komponenten, unter anderem die Vorverarbeitung, beim Systemstart automatisch ebenfalls starten. Für eine detailliertere Beschreibung rund um die Servicedatei und wie das Init-System konfiguriert ist siehe Abschnitt 10.5.

5.3 Verarbeitungsschritte

5.3.1 Validating

Der Verzierungsdienst unterstützt nicht alle existierenden Dateitypen für Keksverzierungsgrafiken, deshalb muss die Korrektheit des Dateityps überprüft werden. Im Umfang dieses Projekts fiel die Wahl in der Planung auf das Dateiformat *SVG* (siehe Abschnitt 3.4).

Geprüft wird das Dokument automatisch in dem Moment, in dem es von der verwendeten Softwarebibliothek *Apache Batik* in ein objektbasiertes Modell verarbeitet wird. Ist die Verzierungsgrafik fehlerhaft, schlägt die Umwandlung fehl und die Softwarebibliothek meldet einen Fehlerfall. Gegenüber der Prüfung gegen ein Referenzdokument des Datentyps SVG (etwa einem XML-Schema) bietet dieser pragmatische Ansatz Geschwindigkeitsvorteile.

Enthält der Upload syntaktische oder inhaltliche Fehler, darf die Verzierung nicht weiter verarbeitet werden. Es wird die Verarbeitung abgebrochen und das Antwortdokument formuliert, in dem von einem Fehlerfall berichtet wird (siehe Unterabschnitt 5.2.1). Das HTTP-Backend reagiert durch eine Fehlerausgabe, die den Kunden zur Korrektur seiner Verzierungsgrafik auffordert.

5.3.2 Filtering

Der SVG-Standard definiert einen sehr grossen Funktionsumfang, doch nicht alle Funktionen werden unterstützt. Einerseits, weil sie für die Keksverzierung keinen Sinn ergeben, andererseits weil im vorgegebenen Zeitrahmen nicht alle durchaus sinnvollen SVG-Funktionen umgesetzt werden konnten. Zum Beispiel finden Animations-Anweisungen beim Kekse verzieren keine Anwendung. Um dem Kunden in diesem Fall kein falsches Bild zu vermitteln, müssen nicht-unterstützte Anweisungen aus dem Verzierungsdokument ausgefiltert werden.

Element	**unterstützte Attribute**	**Notizen**
<svg>	keine	nur als Wurzelelement unterst., verschachtelt nicht möglich
<rect>	x, y, width, height	
<line>	x1, y1, x2, y2	
<polyline>	points	
<polygon>	points	
<circle>	cx, cy, r	
<ellipse>	cx, cy, rx, ry	
<path>	d	alle Subkommandos unterstützt
<text>	x, y, fontSize, font-family	keine Formatierungsanweisungen innerhalb des `<text>`-Elements (z.B. <tspan>), daher wird Text u.a. einzeilig
<g>	keine	Gruppen werden aufgelöst, ansonsten nicht interpretiert

Tabelle 5.1: Unterstützte Funktionen des SVG-Standards

Farbanweisungen werden ausgefiltert, weil es sehr unwahrscheinlich ist, dass ein Glasurmaterial mit genau dieser angegebenen Farbe existiert. Dadurch, dass dem Kunden die Materialwahl für die Glasur freisteht, ist eine Zuordnung von SVG-Farbe zu Material mit ähnlicher Farbe zu aufwendig. Der Kunde soll die Wahl selbst treffen.

Transform-Attribute werden nicht unterstützt, was dazu führt dass einfache Grafikobjekte wie ein Rechteck nicht rotiert auf dem Keks dargestellt werden können. Der Grund war, dass das Transform-Attribut eine grosse Menge an Komplexität mit sich bringt, denn es sind beliebige Matrix-Transformationen damit möglich. Der programmatische Aufwand für Matrix-Transformationen war im gegebenen Zeitrahmen zu groß.

Wiederverwendbare Elemente bzw. Verlinkungen, denn die verwendete SVG-Softwarebibliothek bietet für die Verarbeitung dieser Funktion keine Hilfestellung. Die Verarbeitung müsste eigenständig programmiert werden, jedoch ist die Funktion nicht trivial umzusetzen. Deshalb konnte die Funktion im Projektrahmen nicht umgesetzt werden.

5.3.3 Ungrouping

In diesem Verarbeitungsschritt werden Gruppen-Elemente (`<g>`) vollständig aufgelöst. Im Antwortdokument der Vorverarbeitung finden sich also keine Gruppenelemente wieder. Gruppenelemente dienen in den Grafik-Editoren meistens nur dem Benutzer des Programmes zur Organisation von Grafikobjekten, haben aber keine Auswirkungen auf die resultierende Grafik. Eine Ausnahme dieser Aussage existiert dann, wenn mit Gruppen-Elementen alle enthaltenen Grafikobjekte auf die selbe Weise transformiert werden, d.h. beim Gruppenelement ist ein `transform`-Attribut gesetzt. Wie in Unterabschnitt 5.3.2 offengelegt wurde, wird das Transformieren von Elementen aber nicht unterstützt. Deshalb hat das Auflösen von Gruppen-Elementen keine Folgen.

5.3.4 Repositioning

Ziel dieses Verarbeitungsschritts ist es, die Keksverzierung im Gesamten so zu verschieben, sodass sie die Achsen berührt. Wenn das gegeben ist, liegt die Keksverzierung an einer standardisierten Position. Andere Verarbeitungsssschritte führen ihre Arbeit erst mit diesem Verarbeitungsschritt korrekt aus, weil sie von der Standardposition ausgehen. Das Ergebnis der Neuausrichtung ist in Abbildung 5.5 beispielsweise abgebildet.

Für die technische Umsetzung muss zunächst herausgefunden werden, wie weit die Verzierung abseits vom Koordinatenursprung liegt. Genauer gesagt werden hier die *Minimumwerte* der Verzierung gesucht. Das sind das globale Minimum in Richtung x und das globale Minimum in Richtung y. Wie diese Koordinaten gefunden werden, ist im Unterkapitel Abschnitt 5.5 nachzulesen. Die Aufgabe wird an eine Hilfsklasse übergeben, weil sie in anderen Modulen wiederverwendet wird.

Im zweiten Schritt wird jede Koordinate jedes Grafikobjekts um den Minimumwert der gesamten Verzierung verschoben.Der Minimumwert wird in jedem Fall subtrahiert, weil so eine zu weit auf der positiven Hälfte liegende Verzierung in Richtung Koordinatenursprung verschoben und eine auf der negativen Hälfte liegende Verzierung in die positive Hälfte verschoben wird. Die Definition der Position von SVG-Grafikobjekten ist abhängig vom Typ und daher bei unterschiedlichen Elementtypen ebenfalls sehr unterschiedlich. Deshalb muss jedes Grafikobjekt speziell modifiziert werden, um die gewünsche Verschiebung zu bewirken. Dieser Transformierungs-Schritt wird an eine Hilfsklasse übergeben, da sie auch beim *Rescaling*-Verarbeitungsschritt verwendet wird. Auf das Transformieren von Grafikobjekten wird im Unterkapitel Abschnitt 5.4 eingegangen.

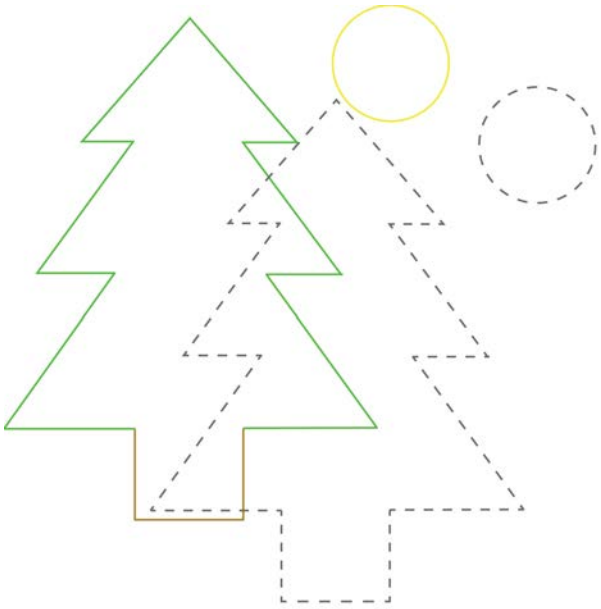


Abbildung 5.5: Eine Keksverzierung wird neu positioniert.
Die Verzierungsgrafik berührt nach der Neuausrichtung beide Koordinatenachsen.
Zuvor wurde die Position vom Kunden für Keksobot unpassend festgelegt.

5.3.5 Rescaling

Als nächster Schritt muss die Grafik skaliert werden, sodass sie innerhalb der Abmessungen des ausgewählten Kekses liegt. Es gibt drei mögliche Stadi in der Beziehung Verzierungsgrafik und Keks: die Grafik ist

- zu groß
- zu klein
- genau passend

Dieser Verarbeitungs-Schritt modifiziert die Verzierungsgrafik nur wenn die Verzierung grösser als der Keks ist. In den anderen zwei Fällen gibt es keine Materialverschwendung bei der tatsächlichen Keksverzierung, weshalb diese Fälle vernachläßigbar sind.

Im ersten Schritt muss beim Rescaling ermittelt werden, wie gross die Verzierung als Ganzes betrachtet ist. Weil zuvor der Schritt *Repositioning* (siehe Unterabschnitt 5.3.4) durchgeführt wurde, liegt die Verzierung auf der positiven Hälfte in der Nähe vom Koordinatenursprung. Aus diesem Grund reicht es aus, die grösste in der Verzierung vorkommende Koordinate zu finden. Dann ist es möglich die Breite (grösste Koordinate x) sowie die Höhe (grösste Koordinate y) der Verzierung zu bestimmen. Wie diese Koordinaten gefunden werden, ist im Unterkapitel Abschnitt 5.5 nachzulesen.

Außerdem ist bekannt wie gross der Keks ist, um dann entscheiden zu können ob eine Skalierung notwendig ist. Im SVG-Dokument für den Keks sind dafür die Felder *width* und *height* des Attributs `viewBox` definiert. Die Felder werden ausgelesen und mit den gefundenen Maximalwerten der Verzierung verglichen. Nur wenn die Verzierung grösser als der Keks ist, ist eine Skalierung notwendig.

Für die Skalierung wird ermittelt mit welchem Faktor multipliziert werden muss, damit die Verzierung innerhalb des Keks liegt. Die Berechnung lautet $Faktor = \frac{Keksmaximum}{Verzierungsmaximum}$.
Wobei das Maximum einmal die Breite (Skalierungs-Schritt 1) und einmal die Höhe (Skalierungs-Schritt 2) ist, je nachdem welcher Faktor gerade berechnet wird.
Dieser berechnete Faktor wird dann auf jede Koordinate in der Verzierungsgrafik angewendet. Weil SVG-Grafikobjekte an unterschiedlichen Stellen bzw. mit anderslautenden Attributen die Koordinaten angeben, wurde diese Aufgabe an eine wiederverwendbare Komponente übergeben. Die programmatische Umsetzung ist in Abschnitt 5.4 beschrieben.

5.4 Hilfskomponente "Transformer"

Mit den Transformierungs-Klassen können Koordinaten und die unterstützten Längenangaben (`width`, `height`, `Radius`) von Grafikobjekten mit einem beliebigen Transformations-Wert addiert, subtrahiert und multipliziert werden. Diese Komponente wird in den Schritten *Repositioning* und *Rescaling* der Verzierungsgrafik verwendet.

Die Modifikation des Grafikobjekts, um die Transformierung durchzuführen, ist sehr vom Typ des Grafikobjekts abhängig. Das Package `svg.stage0.transform` beinhaltet eine Subklasse für jedes unterstützte Grafikobjekt im SVG-Standard. Für Rechtecke beispielsweise werden die Attribute `x` und `y` verändert, allerdings lauten die vergleichbaren Attribute bei Kreisen `cx` und `cy`.

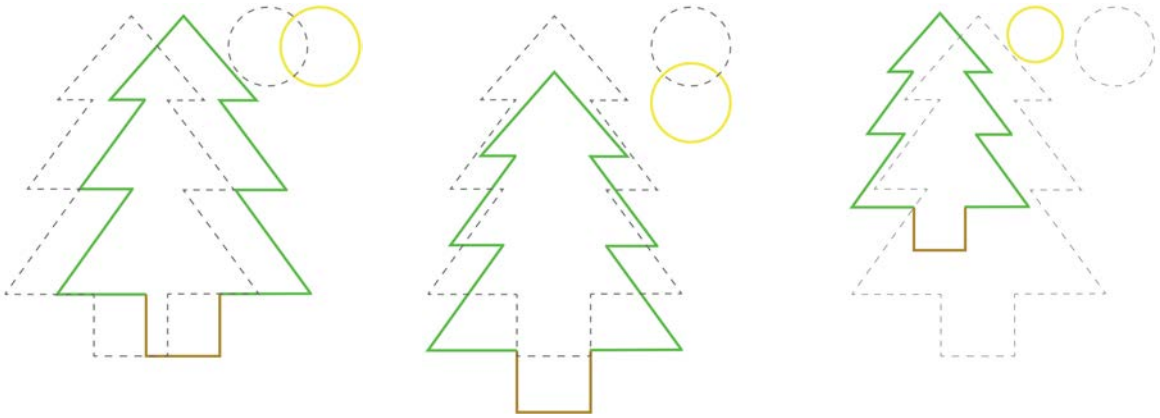
Noch spezieller sind `<path>`-Elemente. Nicht jedes Subkommando im Datenattribut des Pfadelements ist gleich aufgebaut: es gibt Unterschiede in Anzahl von Argumenten und an welcher Stelle Koordinaten angegeben werden. Glücklicherweise bietet die verwendete SVG-Softwarebibliothek *Apache Batik* hier eine komfortablere Lösung für Subkommanods in Pfaden, bei der ein Parser dieser Bibliothek überschriebene Methoden in der Klasse `PathSubcommandTransformer` aufruft.

Es werden die Original-Koordinaten ausgelesen, die Koordinate transformiert, die Attribute abhängig vom Grafik-Objekttyp neu gesetzt und das transformierte SVG-Grafikobjekt an den `RescaleController` zurückgeliefert. Bei Pfad-Elementen findet ein Zwischenschritt des erneuten Verpackens der Subkommando-Objekte in ein Pfad-Element statt.

5.5 Hilfskomponente "Extremwertsuche"

Diese Hilfskomponente findet für ihre Benutzer die Koordinaten in Extrempositionen, also das Minimum und das Maximum aller der Hilfskomponente übergebenen SVG-Grafikobjekte. Sie wird beispielsweise von den Komponente *Repositioning* und *Rescaling* verwendet, damit diese ihre Hauptaufgabe erfüllen können. Eine grafische Veranschaulichung der Arbeit, die diese Hilfskomponente übernimmt, ist in Abbildung 5.8 ersichtlich.

Die `AllDrawingSVGElementHandler`-Klasse verwaltet zu jedem Zeitpunkt Variablen, die den aktuell gefundenen Minimal- sowie Maximalwert für die Achsen `x` und `y` angibt. Die Komponente erhält als Input seiner `handle`-Methoden ein Grafikobjekt und prüft dessen Koordinaten gegen die aktuellen Minimal- und Maximalkoordinaten. Wenn der Input grösser bzw. kleiner ist, wird das aktuelle Maximum bzw.



(a) in x-Richtung verschieben **(b)** in y-Richtung verschieben **(c)** proportional skalieren

Abbildung 5.6: Ergebnisse diverser Transform-Operationen auf eine Verzierungsgrafik

Minimum überschrieben. Das zugehörige Klassendiagramm kann dem Anhang in Abbildung 12.1 entnommen werden.

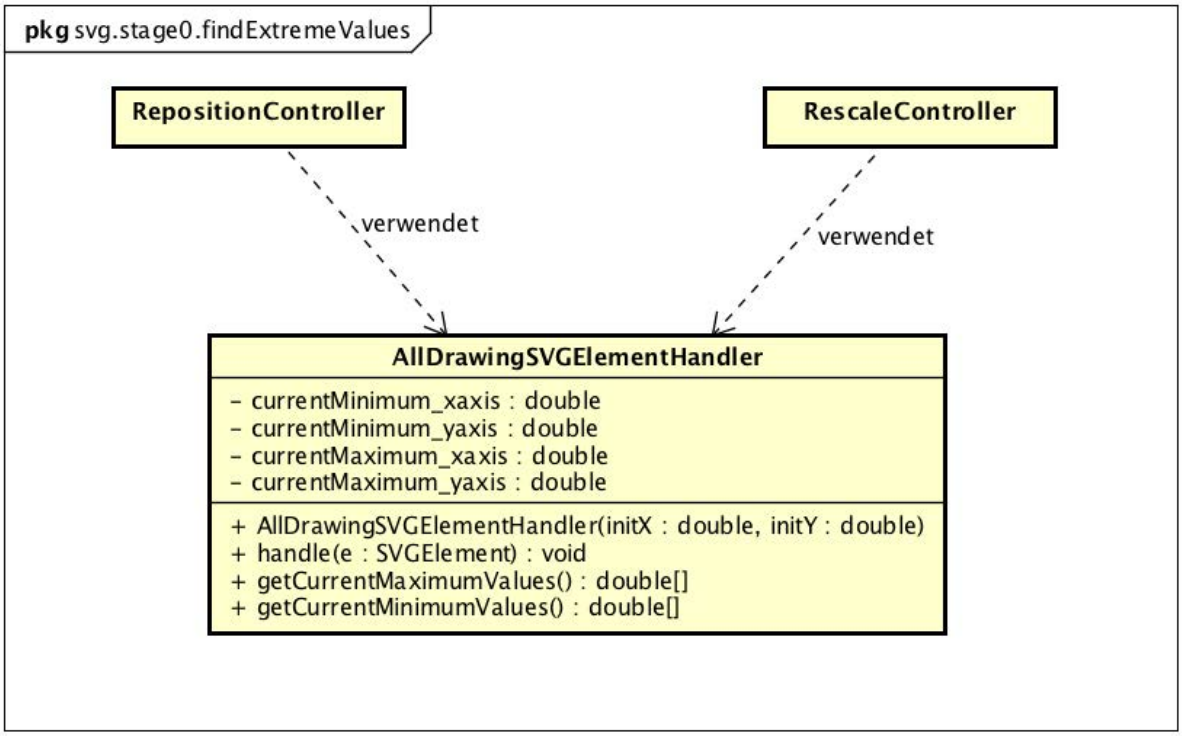


Abbildung 5.7: Verwendung der AllDrawingSVGElementHandler-Klasse

Der Nutzer dieser Hilfskomponente (= andere Verarbeitungs-Komponente) erzeugt eine Instanz von `AllDrawingSVGElementHandler`, um sie zu verwenden. Bei der Instanzierung werden vom Nutzer Koordinaten angegeben, die in der Verzierung tatsächlich existieren: eine Koordinate `x` und eine Koordinate `y`. Das ist notwendig, damit die technische Umsetzung der Hilfskomponente korrekt arbeitet (Vorgangsweise: Vergleiche Koordinaten des Input-Grafikobjekts mit den aktuell in der Keksverzierung als Minimum bzw. Maximum bezeichneten Koordinaten). Der Benutzer sendet alle seine in den Vergleich einzubeziehende Grafikobjekte an die `handle`-Methode dieser Klasse. Nun wird eine für diesen SVG-Elementtyp passende Vergleichs-Klasse instanziert, die die Koordinaten des Inputs mit den gerade aktuellen Extrema vergleicht. Gibt das Element ein neues Maximum oder ein neues Minimum an, wird das zugehörige globale Extremum in `AllDrawingSVGElementHandler` überschrieben. Anschliessend ruft der Benutzer das Ergebnis, also das aktuelle Maximum oder Minimum, von der `AllDrawingSVGElementHandler`-Instanz ab. Die Klassenstruktur von `AllDrawingSVGElementHandler` wird in Abbildung 12.1 beschrieben.

```
public class AllDrawingSVGElementHandler {

    private double currentMinimum_xaxis, currentMinimum_yaxis, currentMaximum_xaxis,
         currentMaximum_yaxis;

    /**
     * Initializes the maximum and minimum trackers for a decoration.
     * The behaviour of setting the trackers to 0 by default leads to no minimum values found
     * if decoration is only on positive side (x > 0, y > 0), because the initial value "0" is the
          smallest coordinate of the decoration.
     * @param initX   Used to initialize the min and max tracker variables for x coord.
     * @param initY   Used to initialize the min and max tracker variables for y coord.
     */
    public AllDrawingSVGElementHandler(double initX, double initY) {
        this.currentMinimum_xaxis = initX;
        this.currentMinimum_yaxis = initY;
        this.currentMaximum_xaxis = initX;
        this.currentMaximum_yaxis = initY;
    }

    public void handleRectangle(SVGElement e) {
            /* extract 'x', 'y', 'width' and 'height' from graphic object (not shown) */

            /* compare x coordinate with current max/min */
            // NOTE:
            // rect_xcoord and rect_ycoord are always defined as "coordiantes nearest to axis
                 origin" (0,0)
            // so if 'rect_xcoord + rect_width' is bigger than current maximum, rect_xcoord is also
            // and if 'rect_ycoord + rect_height' is bigger than current y maximum, then
                 rect_ycoord is too
            if(rect_xcoord+rect_width > this.currentMaximum_xaxis)
                    this.currentMaximum_xaxis = rect_xcoord + rect_width;

            if(rect_xcoord < this.currentMinimum_xaxis)
                this.currentMinimum_xaxis = rect_xcoord;

            /* compare y coordinate with current max/min */
            if(rect_ycoord+rect_height > this.currentMaximum_yaxis)
                    this.currentMaximum_yaxis = rect_ycoord + rect_height;

            if(rect_ycoord < this.currentMinimum_yaxis)
                this.currentMinimum_yaxis = rect_ycoord;
    }

    ...
}
```

Listing 5.4: Beispiel der Extremwert-Suche: Ein Rechteck wird mit den aktuellen Extrema verglichen

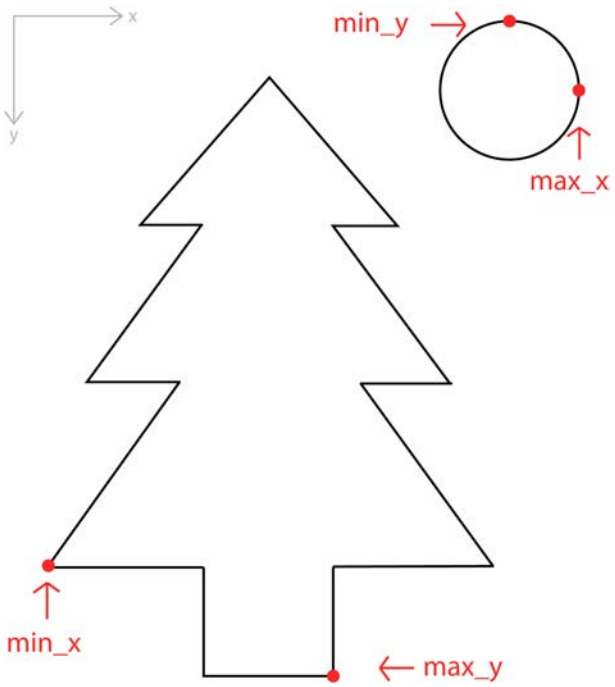


Abbildung 5.8: Extremwerte einer Verzierungsgrafik

5.6 Error Handling & Logging

5.6.1 Error Handling

Die Arbeit des HTTP-Backends mit der Grafikverarbeitung, bzw. die Arbeit des Vorarbeiters selbst, kann zu Fehlern führen. Grundsätzlich wird unterschieden zwischen

- Fehler, die durch ungültige Benutzereingaben entstehen (modelliert als `KbotUserInputException`)
- Netzwerk-Fehler am Weg zwischen Kunde und Grafikverarbeitung (`KbotApplicationException`)
- Fehler, die durch Applikation-Fehlverhalten entstehen (`KbotApplicationException`)

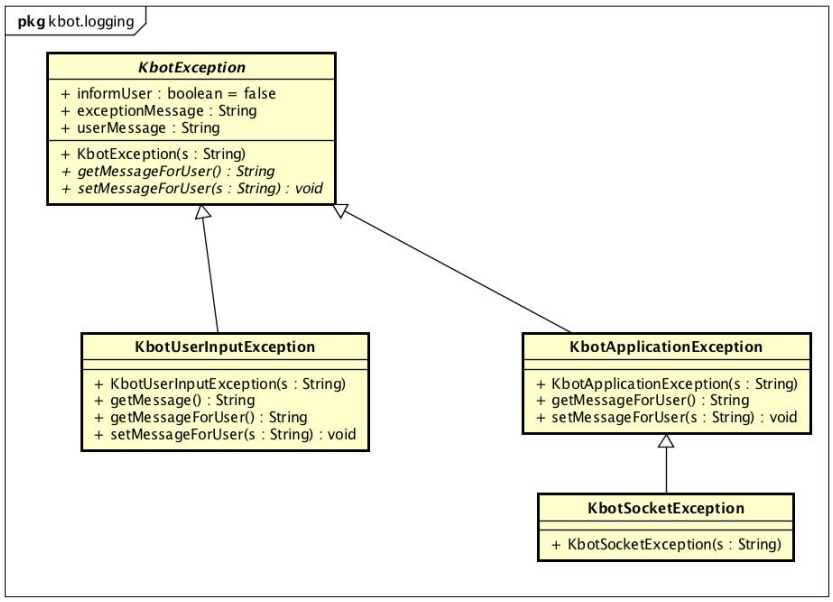


Abbildung 5.9: Exception-Klassen von Keksobot im Überblick

So unterschiedlich die Fehler sind, so unterschiedich sind auch die Massnahmen, die beim Auftreten eines Fehlertyps getroffen werden.

Fehlerbeschreibung	**Maßnahme**	**Auswirkung für d. Nutzer**
Syntaktischer Fehler im User-Upload	Erkennen, Abfangen, Feedback an Nutzer	Fehlermeldung im Bestellvorgang (Schritt 1), Möglichkeit der Korrektur
User-Upload: Falscher Datentyp	—"—	—"—
Netzwerk- bzw. Socket-Fehler	Verbindungsabbruch zum Kunden, Programm nimmt weiterhin andere Anfragen an, Lognachricht vom Level *Warnung*	Umleitung von Bestellvorgang zu Fehlerseite "Bitte später erneut zu versuchen"
Unbekannter bzw. unerwarteter Fehler	Lognachricht vom Level *Kritisch*	Umleitung von Bestellvorgang zu Fehlerseite "Bitte später erneut zu versuchen"

Tabelle 5.2: Fehlervarianten in der Grafikverarbeitung

In der Applikation werden Ausnahmefälle (Java-`Exceptions`) an der Stelle behandelt, an der die genaueste Fehlermeldung definiert werden kann. Eine Hilfsklasse für Socket-Kommunikation beispielsweise hat durch ihre abstrakte Definition nicht die Möglichkeit, eine passende Lognachricht zu formulieren, weil ihr nicht bekannt ist, wie sie konkret in der Applikation eingesetzt wird. In diesem Fall wird die Exception "nach oben" an die aufrufende Klasse weitergegeben, die eine aussagekräftige Beschreibung des Fehlers geben kann.
Im Falle einer unbekannten bzw. in der Applikation nicht erwarteten `Exception` wird diese von der eigenen Klasse `KbotUncaughtExceptionsHandler` abgefangen. Der Grund liegt darin, dass damit der unbekannte Fehler mit der Wichtigkeitsstufe `SEVERE` (siehe Unterabschnitt 5.6.2) aufgezeichnet werden kann. Die Java-Applikation wird konfiguriert, sodass sie unbehandelte `Exceptions` an die eigene Klasse

weiterleitet (siehe Listing 5.5).

```
public class DecorationPreprocessingRequestHandler extends AbstractDaemonRequestHandler {

  @Override
  public void run() {
    /** first and foremost, set a default handler for surprising Exceptions */
    try {
      Thread.setDefaultUncaughtExceptionHandler(new KbotUncaughtExceptionsHandler(this));
    } catch(SecurityException e) {
      System.err.println("cannot instanciate the Exception Handler for Keksobot (using default now)!
          ");
    }

    ...
  }

...
}


public class KbotUncaughtExceptionsHandler implements UncaughtExceptionHandler {

  private static final Logger LOGGER = KbotLogging.getLogger(KbotUncaughtExceptionsHandler.class);

  @Override
  public void uncaughtException(Thread t, Throwable e) {
    ...

      // exception not wrapped in a Kbot exception (= unknown, unexpected)

      // log to 'ERROR' level
      LOGGER.log(Level.SEVERE, "Unknown application error", ex);

    ...
  }

...
}
```

Listing 5.5: Unbehandelte Exceptions werden an Keksobot-Handlerklasse weitergeleitet

Siehe auch die Java-Klassenübersicht in Abbildung 5.9.

Damit im Zweifelsfall die Ursache eines Fehlers gefunden werden kann, ist detailliertes Logging notwendig.

5.6.2 Logging

Die Erklärung, wie auf das Logbuch der Grafikverarbeitungs-Komponente zugegriffen wird, wird in Unterabschnitt 10.9.3 gegeben. Es folgt die technische Dokumentation des Loggings für diese Komponente.

Als Logging-Technologie wurde das in Java SE nativ verfügbare `java.util.logging` eingesetzt. Für die Grafikverarbeitung wurde die flexible Konfiguration des Nachrichtenfilters (Wichtigkeitsstufen bzw. *Log-Level*) und des Ausgabeorts verwendet.

Loggingnachrichten werden je nach Konfiguration der `java.util.logging.Logger`-Instanz zum Beispiel an die Konsole gesendet, in eine Datei geschrieben, oder von einem anderen `java.util.logging.Handler` behandelt. Für die Grafikverarbeitung ist in der Produktionsumgebung das Logging in eine Datei sinnvoll, während des Testbetriebs aber zusätzlich auch die Ausgabe auf die Konsole. Die Kombination ist durch mehrfachen Aufruf der `addHandler()`-Methode des Loggers möglich.

Die Klassen der Grafikverarbeitungs-Komponente erhalten Zugang ihrem Logger, nachdem die `Daemon`-Klasse (mit `main(String[])`-Methode) die Subklasse `KbotPreprocessorLogger` der Klasse `KbotLoggingInstance` initialisiert hat (siehe Listing 5.6). Das macht es möglich, dass alle aufzeichnenden Java-Klassen die selbe, an einer zentralen Stelle (dem `Daemon`) konfigurierbare, `Logger`-Instanz verwenden. Das selbe Vorgehen wird auch für die Keksobot-Komponenten *Order Converter* und *Order Executor* verwendet — dann aber mit einer anderen Subklasse von `KbotLoggingInstance` (Abbildung 5.10).

```
public class DecorationPreprocessingDaemon {
...
  public static void main(String[] args) {
    /* set up logging environment */
    KbotPreprocessorLogger log = new KbotPreprocessorLogger();
    try {
      log.initialize();
    } catch(SecurityException e) {
      System.err.println("Cannot initialize logger! No permission.");
      System.exit(-1);
    } catch(IOException e) {
      System.err.println("Cannot initialize logger! I/O Error.");
      System.exit(-1);
    }

    ...
  }
...
}

public class KbotPreprocessorLogger extends KbotLoggingInstance {
  @Override
  public void initialize() throws SecurityException, IOException {
    Logger l = Logger.getLogger("kbot-preprocessor");
    String path = KeksobotPropertiesFile.getPropertiesObject().getProperty("KBOT_LOG_TARGET");

    if(!Files.isDirectory(Paths.get(path))) {
      throw new KbotApplicationException("Log target defined in Keksobot PROPERTIES file is not a
          directory!");
    }

    FileHandler fh = new FileHandler(path+File.separator+"kbot-preprocessor.%u.log", true);
    fh.setFormatter(new SimpleFormatter());
    fh.setLevel(Level.FINEST);
    l.setLevel(Level.FINEST);
    l.addHandler(fh);
    l.addHandler(new ConsoleHandler());

    setLogger(l);
  }
}

public class RescaleController extends GeneralDecorationPreprocessor{
  private static final Logger LOGGER = KbotLoggingInstance.getLogger(RescaleController.class);

  ...
}
```

Listing 5.6: Initialisierung des Vorverarbeitung-Loggers

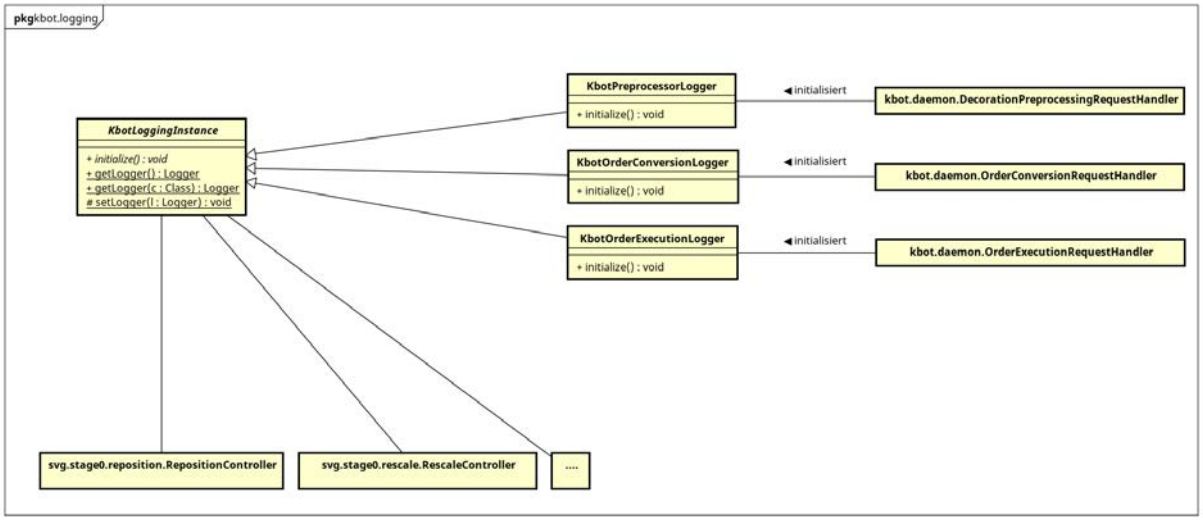


Abbildung 5.10: Klassenübersicht der Keksobot-Logger

Die Wichtigkeits-Stufen der verwendeten Logging-Technologie werden dafür eingesetzt, um wichtige von nebensächlichen oder temporären Fehlern zu unterscheiden. Um tatsächlich bei Bedarf den Ablauf des Programms nachvollziehen zu können, werden auf Wichtigkeits-Stufen unter `INFO` auch Zwischenergebnisse aus Berechnungen (z.B. für das Skalieren der Skalierfaktor) mitgezeichnet.

Stufe	**Verwendungszweck in der Grafikverarbeitung**
`SEVERE`	Kritische Fehler der Applikation (z.B. erkanntes inkorrektes Verhalten)
`WARNING`	Aufgetretene ungünstige Ereignisse, möglicherweise Applikationsfehler (z.B. ordnungsgemäßer Verbindungsabbau fehlgeschlagen)
`INFO`	Benachrichtigung bei üblichen Ereignissen (z.B. nach Akzeptieren neuer Aufträge)
`FINE`	Detaillierte Benachrichtigungen (z.B. beim Betreten einer Teilkomponente des Programms)
`FINEST`	Detaillierte Informationen innerhalb Teilkomponenten (z.B. Berechnungen, Zwischenergebnisse, Statusmeldungen)

Tabelle 5.3: Bedeutung der Wichtigkeitsstufen bei der Grafikverarbeitung

```
KBOT_LOG_TARGET=~/keksobot/logs
KBOT_LOG_MINIMUMLEVEL=FINEST
```

Listing 5.7: Logging-Einstellungen in Keksobot.properties

Kapitel 6

Vom Verzierungsauftrag zum roboterneutralen Format (RG)

6.1 Einführung

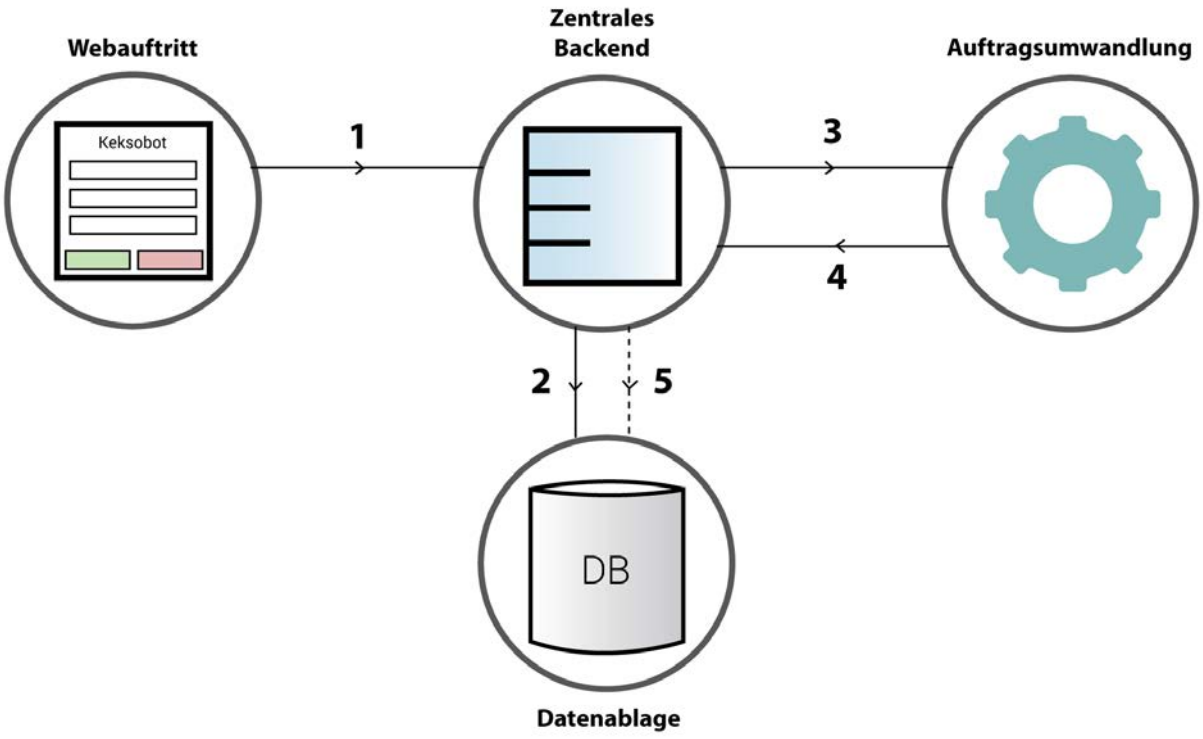


Abbildung 6.1: Datenfluss zwischen Webshop, HTTP-Backend und Keksverzierungs-Umwandler

1. Der Kunde sendet seine Bestellung im Webshop ab
2. Das HTTP-Backend nimmt den Auftrag in der Datenbank auf
3. Das HTTP-Backend leitet den Auftrag zur Konvertierung an den Keksverzierungs-Umwandler
4. Der Keksverzierungs-Umwandler sendet ein *Keksobot Neutral Robot Format* (Details siehe Abschnitt 6.2) als Ergebnis der Umwandlung
5. Das HTTP-Backend empfängt das Konvertierungsergebnis und speichert es im fehlerfreien Fall zum Auftrag in die Datenbank; im Fehlerfall nicht, sodass der Fehler bemerkt werden kann

Mit der Umwandlungs-Komponente wird aus dem aufgenommenen Kundenauftrag ein Dokument vom Format *Keksobot Neutral Robot Format* erzeugt. Das Dokument beinhaltet Auftragsinfos (ID, Stückzahl

Kekse) und Bewegungsanweisungen, die robotertauglich definiert sind. Für den Aufbau des Dokumentenformats siehe Abschnitt 6.2.

Die geometrischen Informationen, die aus der Verzierungsgrafik entnommen werden, müssen in vielen Fällen von grundauf neu definiert werden, damit sie für übliche Robotertypen abfahrbar sind. Ein Kreis zum Beispiel kann nicht als Objekt "Kreis" abgefahren werden: beim Objekt "Kreis" ist kein fixer Start- und Endpunkt für die Bewegung angegeben, weshalb Roboter unendlich viele Möglichkeiten haben, den Kreis abzufahren. Das Objekt "Kreis" ist für Roboter unzureichend spezifisch. Dieses und ähnliche Herausforderungen gibt es für die anderen in SVG möglichen Grafikobjekte. Für alle unterstützten Grafikobjekte in SVG ist eine Umwandlungs-Funktion entwickelt worden.

Das HTTP-Backend verbindet sich zur Umwandlungs-Komponente, nachdem der Kunde einen Auftrag abgesendet hat. Die Konfiguration der Kommunikationsparameter für die Umwandlungs-Komponente ist in Unterabschnitt 10.4.1 beschrieben (Stichwort `KBOT_ORDER_EXECUTION_HOST`).

6.2 Das anlagenneutrale Roboteranweisungs-Dokument

Während der Planung der Durchführung wurde entschieden, dass die Umwandlung von Verzierungsgrafik zu anlagenneutralen Roboteranweisungen in einem selbst-definierten Dokumentenformat endet (siehe Abschnitt 3.5).

Die Anforderungen an das Dokumentenformat sind folgende:

- die drei bei Industrierobotern gängigen Bewegungstypen werden unterstützt: *Punkt-zu-Punkt, Linearbewegung, Zirkularbewegung*
- die Verzierung wird in mehreren Teilmotiven dargestellt: von einander entfernte Figuren werden als eigene Objekt-Einheiten dargestellt
- jedes Teilmotiv enthält die Information über das gewünschte Glasurmaterial
- die Zuordnung eines Anweisungsdokument zum zugehörigen Auftrag ist möglich (Auftrags-ID)
- die Auftragsgröße wird im Dokumentenformat vermerkt
- von Menschen lesbar, d.h. Textformat, verständliche Grammatik und nachvollziehbare Kommandobezeichnungen

Aus diesen Anforderungen ist die Definition des *Keksobot Neutral Robot Format* entstanden. Es ist eine Schemadefinition für das existierende Dokumentenformat Extensible Markup Language (XML). Damit bedient sich das Keksobot-Format der aktuellen Beliebtheit von XML, das in den verwendeten Programmiersprachen gut unterstützt wird. Für die Verarbeitung des Keksobot-Formats ist lediglich eine XML-Verabeitungskomponente notwendig. Die genaue Schemadefinition im XML Schema Definition (XSD)-Dokumentenformat ist in Listing 6.1 einsehbar.

```
<?xml version="1.0"?>
<!--  Keksobot XML document schema, Version 2017-Mar-22 -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Order">  (1)
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ordernr" type="xs:int"/>
```

```
        <xs:element name="cookietypeid" type="xs:int"/>
            <xs:element name="amount" type="xs:int"/>

        <xs:element name="Shape" maxOccurs="unbounded">  (2)
          <xs:complexType>
            <xs:sequence>
              <xs:element name="metadata">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="materialid" type="xs:int"/>  (3)
                    <xs:element name="materialReq" minOccurs="0" maxOccurs="unbounded">  (4)
                      <xs:complexType>
                        <xs:simpleContent>
                          <xs:extension base="xs:string">
                            <xs:attribute name="name" use="required"/>
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>

              <xs:element name="movement" maxOccurs="unbounded">  (5)
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="xstart" type="xs:decimal"/>  (7)
                    <xs:element name="ystart" type="xs:decimal"/>
                    <xs:element name="xend" type="xs:decimal"/>
                    <xs:element name="yend" type="xs:decimal"/>
                    <!--Hilfspunkt für Kreisbewegungen, optional außer bei Kreisbewegungen -->
                    <xs:element name="xcurvemiddle" type="xs:decimal" minOccurs="0"/>
                    <xs:element name="ycurvemiddle" type="xs:decimal" minOccurs="0"/>
                  </xs:sequence>

                  <xs:attribute name="type" use="required">  (6)
                    <xs:simpleType>
                      <xs:restriction base="xs:string">
                        <xs:enumeration value="line"/>
                        <xs:enumeration value="circularArc"/>
                        <xs:enumeration value="point"/>
                      </xs:restriction>
                    </xs:simpleType>
                  </xs:attribute>

                </xs:complexType>
              </xs:element>
              </xs:sequence> <!-- end of Shape -->

              <xs:attribute name="continuousExecution" use="optional" type="xs:boolean"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

Listing 6.1: Schemadefinition des Keksobot Neutral Robot Format

Erläuterungen:

(1) Zu einem Auftrag (`<Order>`) werden *Auftragsnummer, gewählter Kekstyp, Auftragsgröße* gespeichert

(2) Anzahl der Teilmotive der Verzierung (`<Shape>`) ist unbeschränkt

(3) jedes Teilmotiv gibt das ausgewählte *Glasurmaterial* an

(4) das Glasurmaterial des Teilmotives gibt Beschränkungen und verwendete Zusatzfunktionen an

(5) jedes Teilmotiv wird aus einer unbeschränkten Anzahl an Bewegungen definiert

(6) jede Bewegungsanweisung definiert den Bewegungstyp (= einer der drei Grundbewegungstypen)

(7) jede Bewegungsanweisung definiert den Start- und Endpunkt

(8) für Zirkularbewegungen wird zusätzlich ein Hilfspunkt definiert

6.3 Umwandlung von Grafikobjekten

Jedes Grafikobjekt wird mit einer Umwandlungs-Klasse verarbeitet, die für den Typ des Grafikobjekts vorgesehen ist. Die Java-Klasse `kbot.neutraldoc.DecorationNeutralRobotDocumentTranslator` verwaltet hierbei die Umwandlung der Grafikobjekte in das Keksobot-eigene `Shape`-Objekt. Die `Shape`s beinhalten die robotertauglichen Bewegungsdefinitionen und die Materialattribute des für dieses Grafikobjekt verwendeten Glasurmaterials.

Im Verarbeitungsvorgang wird jedes Grafikobjekt innerhalb des SVG-Dokuments auf seinen Typ analysiert und die passende Umwandlungs-Klasse beauftragt. Dadurch, dass alle Umwandlungs-Klassen die Schnittstellen-Definition `svg.objectconverters.GeneralKbotSVGConverter` umsetzen, konnte das Prinzip der Polymorphie angewendet werden, um Programmcode einzusparen und Code-Duplizierung zu verhindern. Die Vorgehensweise der einzelnen Umwandlungs-Klassen wird nun beschrieben.

6.3.1 SVGs "Basic Shapes"

Unter dem Begriff *Basic Shapes* versteht das SVG-Referenzdokument die Elemente [46, Kap. 9]:

- <line>
- <polyline>
- <polygon>
- <rect>
- <circle>
- <ellipse>

Für jedes dieser Elemente steht eine Java-Klasse bereit, die das `GeneralKbotSVGConverter`-Interface implementiert.

Einige Attribute der SVG-Elemente unterstehen speziellen Bedingungen wenn diese unerlaubte Werte haben oder gar fehlen. Diese Bedingungen werden durch das W3C-Referenzdokument in den Kapiteln *Implementation Notes* der Grafikobjekt-Typen vorgegeben. Zum Beispiel führt ein nicht vorhandenes Radius-Attribut in einem `<circle>` dazu, dass der Kreis zu keinen Bewegungsanweisungen führt und somit nicht Teil der finalen Verzierung wird. Diese *Implementation Notes* wurden für die unterstützten Attribute umgesetzt (für die Aufzählung unterstützter Attribute siehe Unterabschnitt 5.3.2).

Linien

Input: Koordinaten x und y für den Anfangs- und Endpunkt der Linie

Verarbeitung: Die Linie wird eins zu eins in eine Linearbewegung übertragen, die Koordinaten dafür werden dem SVG-Element entnommen.

Output: ein Keksobot `Shape`-Objekt mit einer Linearbewegung

```
<line x1="10" x2="30" y1="10" y2="10" />
```

Listing 6.2: SVG-Linie

```
<Shape>
    <metadata> ... </metadata>
    <movement type="line">
        <xstart>10</xstart>
        <ystart>10</ystart>
        <xend>30</xend>
        <yend>20</yend>
    </movement>
</Shape>
```

Listing 6.3: Keksobot-Linie

Polylinien und Polygone

Input: in beiden Fällen eine Liste von Koordinatenpaare fürr die Punkte der Polylinie bzw. des Polygons

Verarbeitung: Die Koordinatenpaare können ohne zusätzliche Aufwände mittels Linearbewegungen übersetzt werden, wobei der vorige Endpunkt der Startpunkt der nächsten Linearbewegung ist. Polygone unterscheiden sich von Polylinien dadurch, dass eine zusätzliche Linearbewegung vom letzten Punkt zum ersten Punkt definiert wird.

Output: ein Keksobot `Shape`-Objekt mit einer Linearbewegung für jede Teillinie der Polyline bzw. des Polygons

```
<polygon points="10,10 20,30 30,10" />
```

Listing 6.4: SVG-Polygon

```
<Shape>
    <metadata> ... </metadata>
    <movement type="line">
        <xstart>10</xstart>
        <ystart>10</ystart>
        <xend>20</xend>
        <yend>30</yend>
    </movement>
    <movement type="line">
        <xstart>20</xstart>
        <ystart>30</ystart>
        <xend>10</xend>
        <yend>30</yend>
    </movement>
</Shape>
```

Listing 6.5: Keksobot-Polygon

Rechtecke

Input: Koordinaten x und y der linken oberen Ecke des Rechtecks, sowie dessen *Breite* und *Höhe*.

Verarbeitung: Das Rechteck wird durch vier Linearbewegungen dargestellt, die jeweils zur benachbarten Ecke führen. Die drei unbekannten Eckpunkte werden über die Angaben der Breite und der Höhe des Rechtecks berechnet.

Output: ein Keksobot `Shape`-Objekt mit vier Linearbewegungen

```
<rect x="10" y="20" width="10" height="10" />
```

Listing 6.6: SVG-Rechteck

```
<Shape>
    <metadata> ... </metadata>
    <movement type="line">
        <xstart>10</xstart>
        <ystart>20</ystart>
        <xend>20</xend>
        <yend>20</yend>
    </movement>
    <movement type="line">
        <xstart>20</xstart>
        <ystart>20</ystart>
        <xend>20</xend>
        <yend>40</yend>
    </movement>
    <movement type="line">
        <xstart>20</xstart>
        <ystart>40</ystart>
        <xend>10</xend>
        <yend>40</yend>
    </movement>
    <movement type="line">
        <xstart>10</xstart>
        <ystart>40</ystart>
        <xend>10</xend>
        <yend>20</yend>
    </movement>
</Shape>
```

Listing 6.7: Keksobot-Rechteck

Kreise

Input: Koordinaten x und y des Kreismittelpunktes, außerdem der Radius des Kreises.

Verarbeitung: Der Kreis wird in zwei Kreisbögen geteilt, wobei deren Hilfspunkt für die Kreisbögen durch Addition (für Punkt `EP2`) bzw. Subtraktion (für Punkt `EP1`) des Radius zur y-Koordinate des Kreismittelpunktes ermittelt wird (die x-Koordinate ist gleichbleibend).

Output: ein Keksobot `Shape`-Objekt mit zwei Kreisbögen

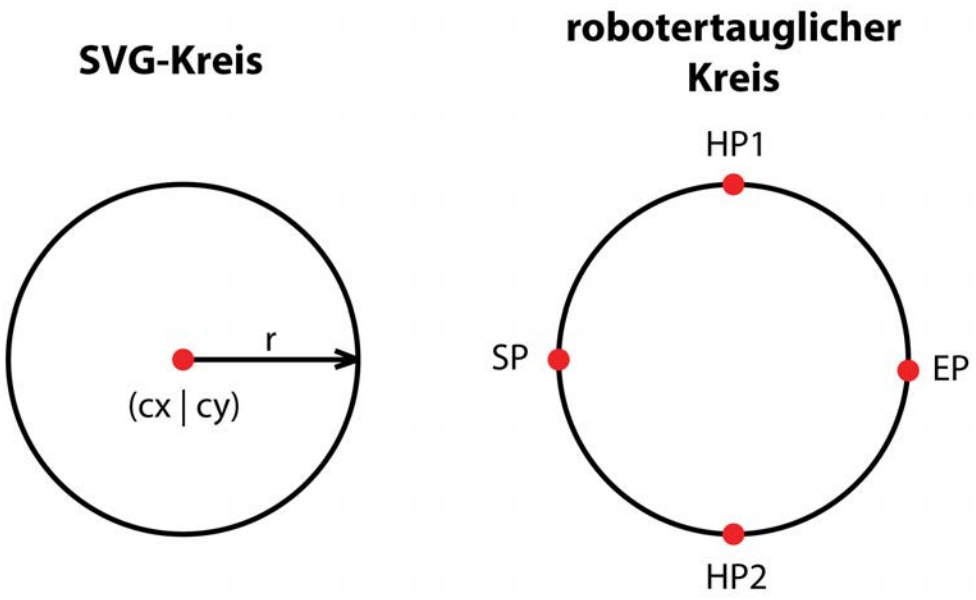


Abbildung 6.2: Umwandlung von SVG-Kreis zu robotertauglichem Kreis

Ellipsen

Input: die Koordinaten x und y des Ellipsenmittelpunkts, außerdem die zwei Radien für die Ellipse.

Ellipsen können in vier elliptische Bögen getrennt werden. Diese werden analog zu elliptischen Bahnen aus `<path>`-Elementen durch kurze Linien angenähert (für Details siehe Unterabschnitt 6.3.2).

Die vier Ellipsenbahnen, die aus der Ellipse stammen, sind wiefolgt definiert (visualisiert in Abbildung 6.3):

- von Punkt $A(c_x - r_x, c_y)$ bis $B(c_x, c_y - r_y)$
- vom Punkt $B(c_x, c_y - r_y)$ bis $C(c_x + r_x, c_y)$
- vom Punkt $C(c_x + r_x, c_y)$ bis $D(c_x, c_y + r_y)$
- vom Punkt $D(c_x, c_y + r_y)$ bis $A(c_x - r_x, c_y)$

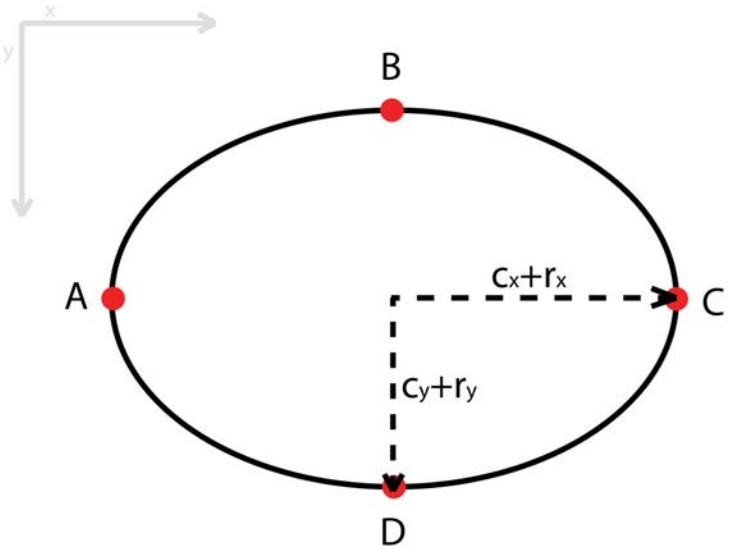


Abbildung 6.3: Ellipse geteilt in vier Ellipsenbahnen

Damit die Ellipse auf die selbe Weise wie elliptische Bögen umgewandelt werden kann, muss dem Umwandler zusätzliche Information übergeben werden. Das ist deshalb so, weil für elliptische Kreisbögen die Angabe des Ellipsenmittelpunktes nicht notwendig ist. Deshalb gäbe es ohne diese Zusatzparameter mehrere mögliche Bögen. Die korrekten Zusatzinformationen für Ellipsen, die durch elliptische Bögen dargestellt werden, sind:

- Rotation: immer gleich 0, weil das `transform`-Attribut von der Grafikverarbeitungs-Komponente ausgefiltert wird
- *large arc flag*: immer `0`, weil der Winkel eines Kreisbogens beim Teilen der Ellipse in vier Kreisbögen nie grösser 180 Grad ist
- *sweep flag*: ein Test hat ergeben, dass der Wert immer `1` ist

Output: ein Keksobot `Shape`-Objekt mit vielen kurzen Linearbewegungen (elliptische Bahnen werden angenähert)

6.3.2 `<path>`-Elemente

Die Umwandlung von Path-Elementen aus SVG ist speziell, weil in einem Path-Element viele Subbefehle enthalten sein können, die eine nicht-festgelegte Menge an Roboterbewegungen ergeben. Im Vergleich dazu ist bei den "Basic Shapes" die Anzahl an Bewegungen vorhersehbar.

Ein Subbefehl ist zum Beispiel eine Linie, eine Kurve, ein elliptischer Bogen, oder eine der anderen erlaubten Subbefehle von Path-Elementen, die in diesem Kapitel beschrieben werden. Die Subbefehle eines Pfades sind miteinander verkettet. Das heißt, das Ende des vorigen Subbefehls ist der Anfang des nächsten. Bei einem "Pfad-Teilstück" handelt es sich um das Resultat aller Subbefehle, die zwischen zwei `move-to`-Subbefehlen stehen. So besteht in einem einzelnen `<path>`-Element das erste Pfad-Teilstück zum Beispiel aus zwei Linien und einer Kurve, wird von einem "Pinsel heben"-Subbefehl getrennt und mit dem zweiten Pfad-Teilstück fortgesetzt.

Abbildung 6.4: Ein `<path>` mit zwei getrennten Teilstücken

Generell hält sich der `PathConverter` auch an ein gemeinsames Interface, so wie andere Konverterklassen das Interface `GeneralKbotSVGConverter` implementieren. Weil aber wie eingangs beschrieben in einem Path-Element mehrere räumlich getrennte Pfad-Teilstücke beschrieben werden können, erzeugt der PathConverter potenziell mehrere `Shape`-Objekte. Das kommt daher, weil bei `<path>`-Elementen räumlich voneinander getrennte Teilpfade möglich sind. Es handelt sich dann um ein spezielles `<path>`-Element, bei dem sich innerhalb der Subbefehle der "Pinsel anheben"-Befehl (`move-to`) befindet. Die Keksobot-Umwandlungskomponente erzeugt in solch einem Fall mehrere `Shape`s. Wenn der Pinsel aber zwischendurch nicht gehoben wird, dann nur ein `Shape`. Durch diese Eigenschaft des Path-Konverters wird nebenbei auch sichergestellt, dass zwischen den Teilpfaden keine Glasur austritt.

In einem Path-Elemenent können sich also mehrere Pfad-Teilstücke befinden, und in jedem Teilstück mehrere Subbefehle (z.B. Linien, Kurven, Bögen, ...). Die Subbefehle eines Pfades werden im Attribut `d` angegeben und enthalten den Buchstaben des Subbefehls gefolgt von dessen Parameter (für eine Übersicht siehe [46, Kap. 8.5]). Die SVG-Softwarebibliothek *Apache Batik* übernimmt die Aufgabe, das `d`-Attribut des `<path>` in einzelne Subbefehle zu teilen.
Jeder Subbefehl wird ähnlich wie bei SVG-Elementen auf seinen Typ untersucht und die Parameter des Befehls an einen Übersetzer für genau diesen Subbefehl-Typ übergeben. Die Übersetzer bieten durch die Implementierung der abstrakten Klasse `PathCommandKbotMovementConverter` die Methoden für die Umwandlung (Klassenstruktur siehe Abbildung 6.5). Bei der Umwandlung werden `Movement`-Objekte erzeugt, die Parameter für eine robotertaugliche Bewegung angeben. Die Datenstruktur dieser Objekte ergibt sich aus dem Keksobot XML-Schema (siehe Abschnitt 6.2).

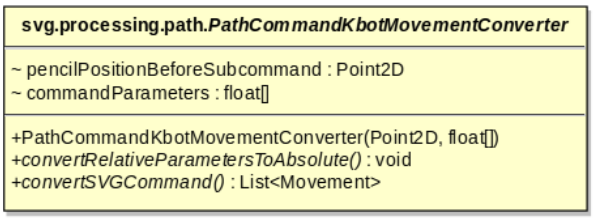


Abbildung 6.5: Schnittstelle für Subcommand-Konverter von Pfaden

Umgang mit relativen Parameterwerten in Subbefehlen

Für beinahe jede Art von Subbefehl von Path-Elementen gibt es relative Versionen, bei denen die im Parameter zufindenden Koordinaten abhängig vom Endpunkt des vorigen Subbefehls sind. Jeder Konverter für Subbefehle muss über das gemeinsame Interface eine Methode implementieren, die relative Parameter mit der Zusatzinfo der aktuellen Pinselposition (Endpunkt des vorigen Subbefehls) zu absoluten Parametern wandeln kann. Diese Funktion wurde in die Konverterklassen ausgelagert, weil nur sie wissen können, welche Parameter absolut gemacht werden müssen und an welcher Stelle sie sich befinden.

Der `PathConverter` ruft die beschriebene *Relativ zu absolut*-Methode wenn notwendig (also wenn Subbefehl eine relative Variante ist) auf. Erst dann findet die Konvertierung statt.

Umwandlung von kubischen und quadratischen Bézier-Kurven

Bézier-Kurven sind Kurven eines bestimmten Grades, die durch eine Anzahl an Punkten, die um Eins höher ist als der Kurvengrad, definiert werden. Bei einer Bézier-Kurve dritten Grades (= kubisch), definieren vier Punkte die Kurve: Startpunkt, Stützpunkt 1, Stützpunkt 2 und der Endpunkt. Start- und Endpunkt sind Teil der Kurve, die Stützpunkte hingegen bestimmen die Krümmung der Kurve und liegen nicht selbst auf ihr (Ausnahmefall: alle Punkte liegen auf einer Linie). Bei quadratischen Bézier-Kurven gibt es hingegen nur einen Stützpunkt. Eine beschriftete exemplarische Bézier-Kurve ist in Abbildung 6.6 sichtbar.

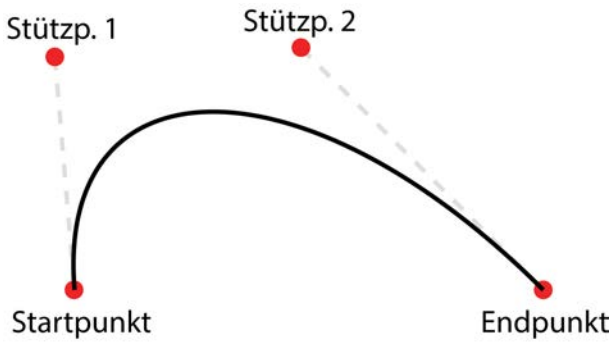


Abbildung 6.6: kubische Bézier-Kurve

Bézier-Kurve gehören bei Industrie-Robotern oft nicht zum Befehlsumfang, so auch beim zur Verfügung stehenden Roboter mit der *KUKA KR C4*-Steuerung. Daher müssen Bézier-Kurven aus dem SVG-Format durch dem Roboter bekannte Bewegungstypen angenähert werden.

Damit die Kurve berechenbar abgefahren wird, kommen für die Annäherung aber von den drei erwähnten Bewegungstypen nur die Linearbewegung und die Kreisbahn in Frage. Bei Punkt-zu-Punkt Bewegungen ist der Bahnpfad des führenden Roboters nicht vorhersehbar. Für Linearbewegungen sowie Kreisbögen existieren Algorithmen zur Annäherung von Bézier-Kurven. Linearbewegungen sind aber wegen ihrer geringeren rechnerischen und geometrischen Komplexität zu bevorzugen. Linearbewegungen werden durch weniger Punkte als Kreisbahnen definiert, denn bei Kreisbahnen gibt es zusätzlich zu Anfangs- und Endpunkt den Hilfspunkt dazwischen.

Um Bézier-Kurven mittels Linearbeweungen anzunähern, gilt es einige Punkte zu finden, die auf der Kurve liegen. Linearbewegungen zwischen den gefundenen Punkten auf der Kurve nähern dann bei genügend hoher Punktedichte die Kurve an. Die Punkte der Kurve lassen sich durch den sogenannten *De Casteljau*-Algorithmus berechnen (beschrieben z.B. in [10]). Die Umsetzung des Algorithmus in Java wurde aus [43] übernommen. Sie wendet den *De Casteljau*-Algorithmus auf kubische Bézier-Kurven an. Als Input benötigt dieser Algorithmus den Start- und Endpunkt der Bézier-Kurve, sowie die zwei Stützpunkte. Der Parameter `t` der De Casteljau-Formel reicht von 0 (= Startpunkt) bis 1 (= Endpunkt). Ein Wert zwischen dem Intervall $[0, 1]$ liefert einen Punkt auf der Kurve.

Der selbe Algorithmus wird allerdings auch für quadratische Bézier-Kurven in der Verzierungsgrafik angewendet: dem Umwandler kubischer Bézier-Kurven wird ein zweiter Stützpunkt mit den selben Koordinaten wie der erste Stützpunkt mitgegeben.

Abbildung 6.7: Bézier-Kurve wird durch Linien angenähert (50 Linien)

"Smooth Cubic Curve" und "Smooth Quadratic Curve"

Es handelt sich um die Subbefehle mit den Buchstaben `S, s, T` und `t`. Sie haben eine Gemeinsamheit, nämlich dass sie jeweils um zwei Parameter weniger benötigen als den Subbefehl, den sie erweitern. Bei Smooth Quadratic Curves muss der einzige Stützpunkt der Kurve (also der den "Hügel" der Kurve bewirkt) nicht mehr angegeben werden. Für Smooth Cubic Curves fällt der erste Controllpoint weg, es bleibt dann nur mehr der zweite.
Das funktioniert deshalb, weil bei den *smooth* Varianten ein Stützpunkt vom letzten Stützpunkt des vorherigen Befehls abhängt. Es handelt sich um eine Reflexion des vorherigen Stützpunkts.

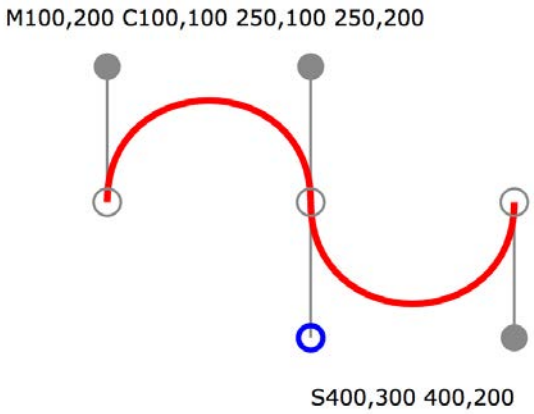


Abbildung 6.8: Die abgebildete "Smooth Cubic Curve" ergibt sich aus dem Stützpunkt der vorigen kubischen Kurve; Bildquelle: `https://www.w3.org/TR/SVG/images/paths/cubic01.svg` (abg. am 16.03.2017)

Um diese Subbefehle umwandeln zu können, speichert der PathConverter den vorherigen umgewandelten Subbefehl temporär in einer Variable. Ist der aktuelle Subbefehl ein solcher *smooth* Befehl, wird aus dem vergangenen Subbefehl der für die Reflektierung veranwortliche Stützpunkt ausgelesen und dem Konverter des *smooth*-Subbefehls übergeben. Mit diesem Stützpunkt rechnet der *smooth*-Subbefehl-Konverter den reflektierten Stützpunkt für seinen Subbefehl aus. Ist das getan, bedienen sich die Konverter von *smoothen* Kurvenvarianten der passenden Standard-Kurvenkonverter (siehe Abschnitt 6.3.2).

Berechnet wird der reflektierte Stützpunkt, indem die Distanz des übergebenen Stützpunkts vom gemeinsamen Punkt (Ende des vorherigen Subbefehls = Start des aktuellen Subbefehls) berechnet und anschliessend in umgekehrter Richtung nochmals angewendet wird. Der gemeinsame Punkt plus der Differenz vom übergebenen Stützpunkt ist also der reflektierte Stützpunkt für den "smoothen" Subbefehl.

Umwandlung von elliptischen Bahnen

Folgende Informationen werden in SVG mit dem Befehl "elliptische Bahn" angegeben:

- Startpunkt der Bahn (= aktuelle Pinselposition)
- Endpunkt der Bahn
- Radius A und Radius B
- Drehwinkel der Ellipse

- "Large arc flag"
- "Sweep flag"

Das "Large arc flag" und das "Sweep flag" geben an, welche Ellipse mit den angegebenen Parametern gemeint ist. Das ist notwendig, weil ansonsten mit den übrigen Parametern mehrere Ellipsen möglich wären.
Mittels des Drehwinkels wird die elliptische Bahn gedreht, nicht aber der gesamte Pfad, den das `<path>`-Element darstellt.

Industrie-Roboter besitzen oft keine eigenen Befehle für elliptische Bahnen, so auch der zur Verfügung gestellte Roboter mit *KUKA KR C4*-Steuerung. Die elliptischen Bahnen werden daher durch andere Bewegungen angenähert. Letztendlich läuft die Umwandlung auf eine Reihe von Linearbewegungen hinaus. Durch Erhöhung der Geradenanzahl wird die Genauigkeit erhöht.

Für die Umwandlung wird ein Algorithmus verwendet, der elliptische Bahnen in Bézier-Kurven dritter Ordnung wandelt [29, S. 18]. Der Algorithmus wurde eigentlich dafür entwickelt, um frei im Koordinatensystem platzierte Ellipsen (d.h. auch rotiert) in definierbarer Genauigkeit für die Darstellung auf zweidimensionale Anzeigemedien mit fixem Pixel-Raster anzunähern (also z.B. einem LCD-Display). Mit diesem Algorithmus und der vorhandenen Logik für die Übersetzung von Bézier-Kurven in eine Reihe von Linearbewegungen können elliptische Bahnen angenähert werden.

Es wurde die Formel für Kurven dritter Ordnung für die Umwandlung von der elliptischen Bahn zur Bézier-Kurve gewählt, weil die Genauigkeit höher als bei einer Kurve zweiter Ordnung ist.
Zum Start der Umwandlung wird zuerst aus dem erhaltenen Input der Mittelpunkt der zugehörigen Ellipse festgestellt (die Vorgehensweise ist in der W3C Spezifikation beschrieben, siehe [46, Kap. 6.5]).

Der Algorithmus zur Umwandlung einer elliptischen Bahn zur Bézier-Kurve dritter Ordnung (siehe [29, S. 18]) benötigt in seiner Hauptformel die Werte des sogenannten Parameter η für den Start- und Endpunkt, weil der Algorithmus von einer parameterisierten Ellipsenfunktion ausgeht. Aus dem SVG-Element ergeben sich aber nur die Koordinaten dieser beiden Punkte. Für die Berechnung des Parameter η steht eine weitere mathematische Funktion $E(\eta)$, zu finden in [29, S. 5], zur Verfügung.
Der Wert der Parameter η für Start- und Endpunkt wird in die Hauptformel eingesetzt. Daraus ergeben sich die Stützpunkte (in der Formel: `Q1` und `Q2`) für die Bézier-Kurve dritter Ordnung, wodurch alle vier Punkte bekannt sind und die elliptische Bahn umgewandelt wurde.
Die Implementierung dieses Verfahrens wurde einem quelloffenen Java-Projekt entnommen [28], das praktischerweise bereits für die Verwendung mit SVG-Elementen ausgelegt ist. Die Java-Klasse wurde minimal angepasst, sodass die Ergebnisse der Berechnung als Rückgabewert zurückgeliefert werden, anstatt an die Konsole gesendet zu werden.

Mit den Parametern der Bézier-Kurve dritter Ordnung wird nun die in Abschnitt 6.3.2 beschriebene Konverterklasse für Bézier-Kurven in Pfad-Elementen aufgerufen.

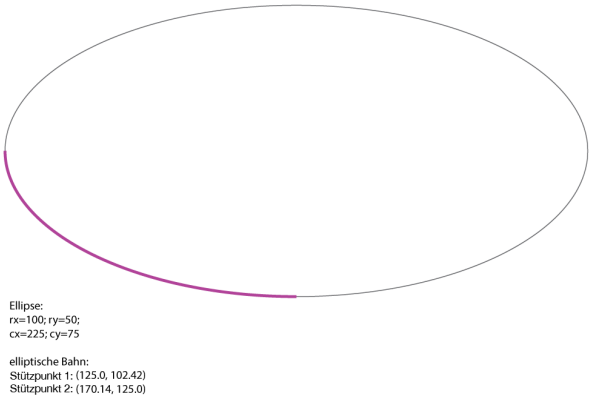


Abbildung 6.9: elliptischer Bogen wird durch kubische Bézier-Kurve angenähert

Linien, horizontale Linien, vertikale Linien in <path>-Elementen

Sie sind sehr einfach definiert, geben im Falle von Linien nur die Koordinaten des Endpunktes an, bzw. bei horizontalen/vertikalen Linien nur die Differenz auf der jeweiligen Achse. Das Ergebnis der Umwandlung dieser Subbefehle ist lediglich ein `Movement`-Objekt.

6.3.3 Umwandlung von <text>-Elementen

Das Text-Element kann als eines der aufwendigst zu übersetzenden SVG-Elemente angesehen werden. Es unterstützt alle auf dem Anzeigegerät installierten Schriftarten, verschiedenste Schriftgrössen und Grösseneinheiten, Veränderungen der Fonteigenschaften (Abstand zwischen Zeilen und Buchstaben), Textorientierung (*left-to-right, right-to-left, horizontal, vertical*), und weitere (siehe [46, Kap. 10]).

Die volle Umsetzung ist im Projektrahmen zeitlich nicht machbar, außerdem wurde vermutet, diese Funktionen werden nur selten verwendet. Aus diesem Grund wurde entschlossen nur einen kleinen Teil der breiten Funktionalität abzudecken und übersetzen zu können. Unterstützt wird:

- simpler, einzeiliger Text (d.h. keine spezielle Formatierung)
- Positionierung des Textes in der Keksverzierung
- Schriftarten (falls diese nicht verfügbar sind, wird eine Standard-Schriftart als Ersatz verwendet)
- Schriftgröße (Angabe erfolgt einheitslos, wird als Höhe der Schrift-“Baseline” in Millimeter interpretiert)

Es ist möglich, sofern die Schriftart im Katalog von Keksobot verfügbar ist, die vom Kunden gewünschte Schriftart mit dem Roboter abzufahren. Wie als Keksobot-Administrator eine Schriftart zum Keksobot-Katalog hinzugefügt wird, kann in Abschnitt 10.8 nachgelesen werden. Unterstützt werden Schriftarten im TrueType-Format.
In jeder TrueType-Schriftart sind die Umrisse für alle in der Schriftart unterstützten Glyphen (Buchstaben, Zahlen, Symbole) definiert.
Das Auslesen einer TrueType-Schriftart, mit dem Ziel die Umrisse zu erhalten, ist nicht ganz einfach. Der TTF-Standard legte den Fokus auf kleinen Speicherverbrauch, weshalb Informationen hinter kryptischen Abkürzungen versteckt und möglichst komprimiert formuliert sind [44, Kap. 6]. Die Python-Softwarebibliothek `ttfquery` (Version 1.0.5) wurde ausgewählt, um mit den Implementierungsdetails des TrueType-Formats umzugehen. Eine vergleichbare Software für die im restlichen Konverter angesetzte

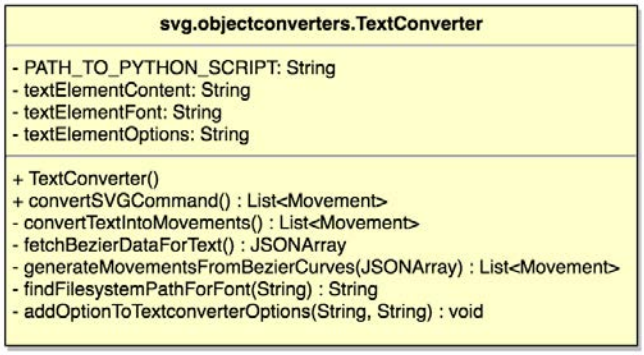


Abbildung 6.10: Text-Converter: Klassendiagramm der Java-Komponente

Sprache Java wurde nicht gefunden. Das bedeutet also, dass der `<text>`-Konverter zum Teil in Java und anderen Teil in Python geschrieben wurde. Die Java-Komponente des `<text>`-Konverters ruft also die Python-Komponente auf, um den Text zu übersetzen. Die Java-Komponente verpackt das Berechnungsergebnis anschließend in Objekte von Keksobot (`Movement` und `Shape`).

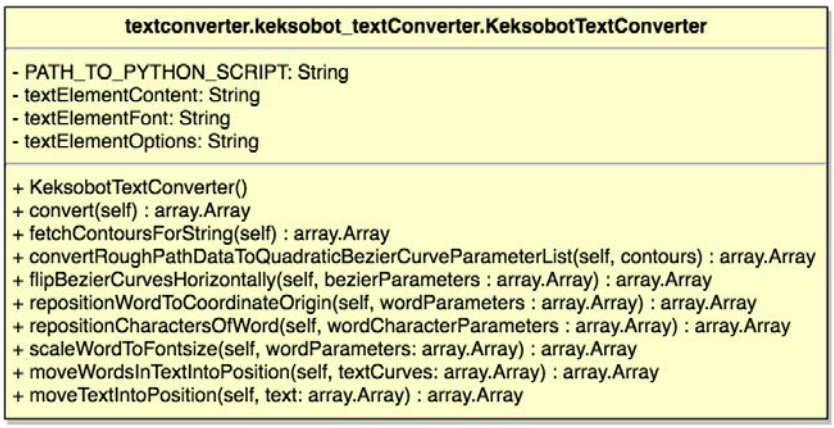


Abbildung 6.11: Text-Converter: Klassendiagramm der Python-Komponente

Die Interaktion zwischen Java- und Python-Komponente ist in Abbildung 6.13 zu sehen. Die Java-Komponente `TextConverter` führt die Python-Komponente über Javas `Process`-Klasse aus. Der Pfad zum Python-Script ist in der Java-Datei hinterlegt, also nicht vom Keksobot-Administrator konfigurierbar. Der Python-Komponente werden *Dateisystem-Pfad zum TTF der verwendeten Schriftart*, *der gewünschte einzeilige Text* und *Zusatzinformationen für den Konverter* als Kommandozeilenparameter übergeben. Unter *Zusatzinformationen* werden *Schriftgröße*, *Position in x*, *Position in y*, *relative Verschiebung des Textes in x und y* angegeben.

In der Java-Komponente wird geprüft, ob die im `<text>`-Element angegebene Schriftart im Schriftarten-Katalog der Keksobot-Installation verfügbar ist. Dafür wird das Attribut `font-family` vom `<text>` ausgelesen und im Schriftarten-Verzeichnis von Keksobot nach einer Datei mit diesem Namen gesucht. Damit das funktioniert, muss eine Schriftart wie in Abschnitt 10.8 für Keksobot-Administratoren beschrieben installiert werden. Technisch bedeutet das bei der Schriftart-Installation, dass der Name der Schriftart aus der gegebenen TTF-Datei entnommen wird, die gegebene Schriftart-Datei auf diesen tatsächlichen Schriftart-Namen umbenannt und in das Keksobot-Schriftartverzeichnis verschoben wird. Dann ist es bei der Verarbeitung von `<text>`-Elementen möglich, nach einer Datei mit dem Namen der Schriftart zu suchen. Gibt es die gewünschte Schriftart nicht, dann wird eine Standard-Schriftart verwendet. Die Standard-Schriftart kann vom Keksobot-Administrator konfiguriert werden, siehe dafür Unterabschnitt 10.4.1. Gibt es die Schriftart hingegen im Katalog, wird ihr Dateisystem-Pfad beim Aufruf der Python-Komponente mitangegeben.

Von der Python-Komponente wird das Konvertierungsergebnis zurück zur Java-Komponente kommuniziert, indem es auf den Stream `STDOUT` geschrieben wird. Im Java-Teil kann dieser Stream ausgelesen werden, dient somit also zum Abrufen des Ergebnisses.
Die Java-Komponente wandelt das als eine Zeichenkette kodierte Konvertierungsergebnis wieder in ein Java-Objekt um. Derzeit wird als Kodierung des Konvertierungsergebnisses die Array-Schreibweise von JavaScript Object Notation (JSON) verwendet, sodass das Format z.B. wie in Abbildung 6.12 aussieht.

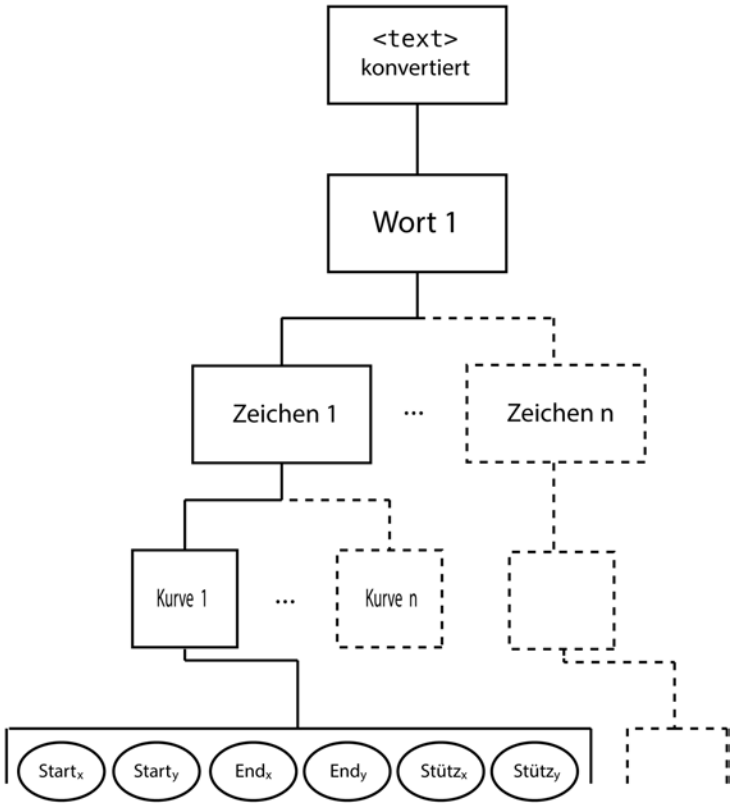


Abbildung 6.12: Kodierung des konvertierten Text-Elements aus Python für Java

Aus den Java-Objekten des Konvertierungsergebnisses werden nun Keksobot `Movements` und `Shapes` erzeugt. Im Konvertierungsergebnis befinden sich die Parameter, die eine quadratische Bézier-Kurve definieren, da bei TTF-Schriftarten die Glyphen als solche Bézier-Kurven angegeben werden [44, Kap. 1]. Die Konvertierungs-Logik für quadratische Kurven wird demnach verwendet, um die `Movement`-Objekte zu generieren. Aus jedem Zeichen aus dem Text entsteht ein `Shape`-Objekt. Das bedeutet aber, dass bei Text akzeptiert wird, dass auch unverknüpfte Bewegungsanweisungen in einem gemeinsamen `Shape`-Objekt enthalten sind. Die anderen Umwandlungs-Komponenten folgen dem Prinzip, dass eine Unterbrechung der Form ein neues, getrenntes `Shape`-Objekt impliziert. Hier wird darauf verzichtet, weil sich die Textumwandlung um den Vergleich *hängt zusammen / hängt nicht zusammen* für jede Kurve jedes Zeichens erweitern würde. Die Umwandlung von Text nimmt dann mehr Ressourcen in Anspruch, aber die Resultate unterscheiden sich nicht. Jedes `Movement` beinhaltet die Startkoordinaten an, zu denen sich der Roboter zuerst, ohne Verzierung aufzutragen, bewegt. Es kommt also durch ein `Shape`-Objekt für ein mehrteiliges Zeichen (z.B. ein Semikolon `;`) zu keinen Fehlverzierungen.

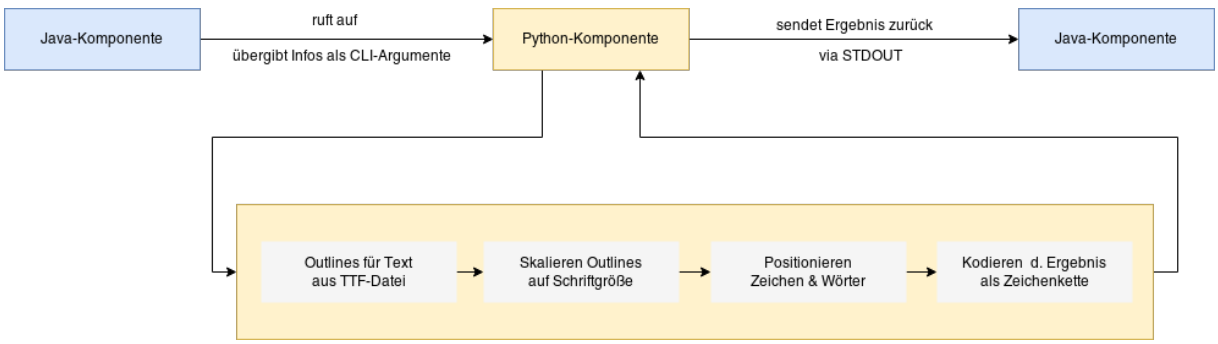


Abbildung 6.13: Text-Converter: Interaktion zwischen Java-Komponente und Python-Komponente

Die Python-Komponente liest die Umrisse der im gegebenen Text enthaltenen Buchstaben aus. Dabei geht leider die für die Schriftart typische Abstand zwischen Zeichen (in der Typografie als *Kerning* bezeichnet) verloren. Die Software-Bibliothek unterstützt das Auslesen des Kerning-Werts nicht. Aus dem Grund muss auf eine eigene Positionierung auf der horizontalen Achse zurückgegriffen werden. Die Buchstaben werden in fixen Abständen, aber abhängig von der Schriftgröße, positioniert. Berechnet wird der fixe Abstand wie folgt:

$$Zeichenabstand = \frac{BreiteeinesSpacecharacters}{4}$$

Die Berechnung entstammt eigener Abschätzungen, wie groß Zeichenabstände in der Schriftart *Arial* gewählt wurden. Der berechnete Abstand wird dann auf die bisherige Position des Zeichens (= an der Kante des vorigen Zeichens) hinzuaddiert, sodass zwischen ihnen ein Abstand liegt.

Bei `<text>`-Elementen, die aus mehreren Wörtern bestehen, müssen die Wörter nach Positionierung ihrer Zeichen ebenfalls in Abständen positioniert werden. Das *Space*-Zeichen deutet auf einen Text mit mehreren Wörtern hin, andere unsichtbare Zeichen werden ignoriert. Der Abstand eines *Space*-Zeichens in der gewählten Schriftart wird abgefragt und damit die Wörter positioniert.

6.4 Berücksichtigung der Eigenschaften von Glasurmaterialien

Nicht jedes Glasurmaterial verhält sich gleich. Es gibt zwischen Glasurmaterialien z.B. Unterschiede in den Fließeigenschaften, die dazu führen, dass das Verzierungsergebnis bei gleicher Verzierung, aber zwei unterschiedlichen Glasurmaterialien, nicht die selbe Qualität hat. Ist ein Glasurmaterial zähflüßiger als das andere, dann muss der Roboterarm in diesem Fall die Verzierung langsamer abfahren, damit der Glasurspender eine durchgängige Verzierung auf den Keks auftragen kann. Ansonsten ist die Gefahr einer rissigen Verzierung gegeben. Die andere Möglichkeit eine rissige Verzierung zu verhindern ist, zähflüssigere Materialien mit höherer Kraft aus dem Glasurspender zu drücken als flüssigere. Das Ziel von beiden Lösungsansätzen ist, die Verzierung in gleichbleibender Qualität, unabhängig vom verwendeten Material, auf dem Keks abzubilden.

Für die Keksverzierung wurde auch der Bedarf berücksichtigt, bei bestimmten Materialien bei der Keksverzierung zusätzliche Werkzeuge aktivieren zu können. Das Werkzeug könnte beispielsweise ein Heizstrahler oder ein Gebläse sein, um das Material zu trocknen. Für Keksobot stellt das ein zusätzliches Attribut eines Materials dar, das im Dateiformat *Keksobot Robot Neutral Format* vorzusehen ist.

Die Produktionsumgebungen, in denen Keksobot verwendet wird, werden sich in der Praxis wahrscheinlich sehr voneinander unterscheiden: unterschiedliche Werkzeugtypen werden eingesetzt, sodass es nicht möglich ist sich in der Software auf fixe Material-Zusatzbedingungen festzulegen. Aus diesem Grund legt der Keksobot-Administrator bei der Materialdefinition die Art des Zusatzattributs und das Attribut ansich eigenständig fest.
Für die prototypische Testproduktion von Keksobot werden zum Beispiel folgende Materialattribute verwendet:

- Geschwindigkeit des führenden Roboters [Einheit: mm/s]
- Volumenstrom des Glasurspenders [Einheit: ml/s]

- Heizungsfunktion (optional angegeben, standardmäßig *aus*)
- Gebläsefunktion (optional angegeben, standardmäßig *aus*)

Bei der Auftragsumwandlung werden die Zusatzbedingungen des verwendeten Materials aus der Datenbank abgerufen und in das Ausgabedokument eingebunden. Für die prototypische Testproduktion sieht das Ausgabedokument deshalb wie folgt aus:

```
<?xml version="1.0" encoding="utf-8"?>
<Order>
    <ordernr>1</ordernr>
    <cookietypeid>1</cookietypeid>
    <amount>1</amount>
    <Shape>
        <metadata>
            <materialid>1</materialid> <!-- Buttercreme natur -->
             <materialReq name="velocity≫2.5</materialReq>
                  <!-- Roboter bewegt sich mit 2.5 mm/s -->
             <materialReq name="dispenser≫1</materialReq>  <!-- Glasurspender tragt 1 ml/s auf -->
             <materialReq name="heating≫false</materialReq>  <!-- Heizung wird nicht benötigt -->
             <materialReq name="fan≫false</materialReq>  <!-- Gebläse wird nicht benötigt --> }
        </metadata>

        <movement type="line">
            <xstart>5.20</xstart>
            <ystart>52.74</ystart>
            <xend>50.69</xend>
            <yend>46.14</yend>
        </movement>

        .... (gekürzt) ....
```

Listing 6.8: Materialattribute im Keksobot Robot Neutral Format

Die Funktionsweise von Glasurspendern kann sehr unterschiedlich sein. Der Glasurspender der Testproduktion drückt das Glasurmaterial beispielsweise mechanisch aus einem Spritzbeutel, ein anderer Glasurspender könnte hingegen mit Druckluft arbeiten. Wie die Angabe "Volumenstrom" in der Produktionsanlage erreicht wird, hängt daher stark von den darin verwendeten Werkzeugen ab. Der mechanische Glasurspender erzeugt aus der Volumenstrom-Angabe den Befehl für die `Motorumdrehungen pro Min.`, der Druckluft-Glasurspender dagegen eine Angabe in z.B. `bar`. Es liegt in der Hand der Produktionsschnittstelle, dieses Materialattribut für den verwendeten Glasurspender zu übersetzen.

Der Kunde hat während des Bestellvorgangs die Möglichkeit, das Glasurmaterial für jeden Teil seiner hochgeladenen oder ausgewählten Keksverzierung festzulegen. Von der Kundenschnittstelle wird die Information an das HTTP-Backend weitergeleitet, wo die Materialwünsche dem Kundenauftrag zugeordnet werden. Erst nach Ende des Bestellvorganges fügt das HTTP-Backend einen neuen Auftrag samt der zugehörigen Daten (Verzierungsgrafik, Materialauswahl, ...) in die in diesem Kapitel beschriebene Umwandlungs-Komponente ein.

Die Umwandlungs-Komponente erhält die Glasurmaterial-Auswahl direkt aus der Verzierungsgrafik, die im Dokumentenformat SVG definiert wird. Der SVG-Standard erlaubt es aufgrund seiner Abstammung aus der Familie der XML-basierten Dokumentenformate, das Dokumentenformat um beliebige Datenelemente zu erweitern. Im Falle von Keksobot wurde dies genutzt, indem zu jedem Grafikobjekt in der

Verzierungsgrafik ein Keksobot-eigenes Attribut mit der eindeutigen Nummer des ausgewählten Materials angegeben wird. Das HTTP-Backend pflegt die Materialauswahl also in das SVG-Dokument der Verzierungsgrafik ein, bevor der Auftrag an die Umwandlungs-Komponente weitergeleitet wird.

Die Umwandlungs-Komponente, die aus dem SVG-Dokument schlussendlich ein anlagenneutrales Roboteranweisungs-Dokument generiert, liest nun jedes Grafikobjekt aus der Verzierung ein, wandelt es in robotertaugliche Bewegungen um und generiert eine Instanz der Keksobot-eigenen Klasse `Shape`.
Zu jedem Shape-Objekt werden die eingangs vorgestellten Materialattribute vermerkt, die auf das ausgewählte Material rückzuführen sind. Sie werden aus der Datenablage gelesen.

Nachdem jedes Grafikobjekt umgewandelt wurde, wird das anlagenneutrale Roboteranweisungs-Dokument verfasst (für dessen Definition siehe Abschnitt 6.2). Das Dokument leitet die Umwandlungs-Komponente über die bestehende Netzwerkverbindung zurück an das HTTP-Backend.

Schlussendlich werden die Materialeigenschaften während des Keksverzierungs-Prozesses berücksichtigt. Die Roboterkommunikations-Komponente sendet die Zusatzbedingungen an die Robotersteuerung, die diese entweder auf sich selber anwendet (*Robotergeschwindigkeit*), oder an das passende Werkzeug weiterleitet. In der Praxis wird die Weiterleitung beispielsweise ein Signal auf einen Ausgang zum gemeinsamen Feldbus sein. Das Werkzeug liegt an diesem Feldbus an, sodass ein Signal von ihm registriert und die Funktion ausgeführt wird (z.B. *Heizung an*).

6.4.1 Ermittlung der Materialattribute

Die Attribute von Glasurmaterialien können durch Materialtests ermittelt werden, die mit unterschiedlichen Parametern so oft wiederholt werden, bis das Keksverzierungs-Ergebnis qualitativ zufriedenstellend ist.

Die Materialattribute *Geschwindigkeit* und *Volumenstrom* hängen auch von den individuellen Bedürfnissen des Produzenten ab. Je schneller der führende Roboter die Verzierung abfahren kann, desto schneller ist der Keks verziert. Der Volumenstrom hat auch Einfluss darauf, wieviel Material aufgetragen und damit verbraucht wird. Wie diese zwei Materialattribute als für die Produktion optimal definiert werden, hängt auf der einen Seite vom Glasurmaterial und dessen Materialeigenschaften (Konsistenz, Fließfähigkeit) ab, aber auch vom Unternehmer, der die optimale Verzierungsqualität definiert.

Die Materialattribute *Heizung* und *Gebläse* beruhen auf den Erfahrungen des Bäckers mit dem Material und die Art und Weise, wie Kekse nach Ende des Verzierungsprozesses weiterbehandelt werden. Es kann hier ebenfalls keine universelle Antwort gegeben werden.

Die Ergebnisse des Materialtests werden schließlich bei der Eintragung eines neuen Glasurmaterials im Keksobot-Webauftritt angegeben. Als Benutzer der Gruppe *Bäcker* kann dieser Vorgang durchgeführt werden. Für die genauere Beschreibung siehe Abschnitt 10.7.

6.5 Error Handling & Logging

In der Umwandlungs-Komponente können im Prozess der Umwandlung Fehler auftreten. Im Gegensatz zur Grafikverarbeitungs-Komponente (siehe Abschnitt 5.6) ist hier allerdings ein Fehler mit der Benutzereingabe (= die Keksverzierung des Kunden) ausgeschlossen, denn die Keksverzierung wurde bereits überprüft und im Falle eines Fehlers die Fortsetzung des Bestellvorganges verhindert. Es wird also zwischen den Fehlertypen *Netzwerktechnischer Fehler* und *Programmfehler während der Umwandlung* unterschieden.

Ein Programmfehler, der während des Umwandlungs-Prozesses auftrat, muss genauer analysiert werden. Ein solcher Fehler sollte im Regelfall nicht auftreten, kann aber aufgrund unvorhergesehener Zustände vorkommen. Wie bei der Fehleranalyse vorgegangen wird, wird in Abschnitt 10.9 beschrieben.

Die Folge eines Programmfehlers ist, dass die Umwandlung des Auftrages nicht abgeschlossen werden kann. Zu diesem Auftrag ist in der Datenbank kein Umwandlungsergebnis und somit auch kein Zeitstempel für Ende der Umwandlung vermerkt. Das wiederum hat zur Folge, dass der Verarbeitungsprozess für diesen Auftrag stillgelegt wird, weil ein fehlerhaft umgewandelter Auftrag nicht weiter an die Auftragsausführung zu senden ist.

Als Administrator in der Keksobot-Benutzerschnittstelle können fehlerhaft verarbeitete Aufträge angezeigt und ein weiterer Umwandlungs-Vorgang gestartet werden. Für Details zur Benutzerschnittstelle für diese Funktion siehe Unterabschnitt 10.9.2.

Um die Ursache eines Programmfehlers zu finden und dabei möglicherweise einen systematischen Fehler zu erkennen, werden Programmmeldungen abgestuft nach ihrer Wichtigkeit aufgezeichnet. Die Programmmeldungen geben Aufschluss darüber, an welcher Stelle im Umwandlungs-Prozess und bei welchem Auftrag der Fehler aufgetreten ist.

6.5.1 Error Handling

Das Java-Programm registriert Ausnahmefälle (in Java `Exceptions`), die an verschiedensten Stellen des Programms auftreten können. Die Ausnahmefälle werden an der Stelle behandelt, an der die spezifischste Beschreibung des Ausnahmefalls gegeben werden kann. Wenn ein Ausnahmefall eintritt, wird eine Programmeldung in der zutreffenden Wichtigkeitsstufe in das Logbuch geschrieben. Ausnahmefälle müssen nicht notwendigerweise auf einen Fehler hindeuten, sondern können Teil des ordnungsgemäßen Programmablaufs sein (z.B. wenn ein pausierter Programmteil von außen zur Fortsetzung aufgefordert wird). Deshalb wird anhand des Typs des Ausnahmefalls entschieden, ob eine Programmmeldung für das Logbuch notwendig ist und welche Maßnahme getroffen wird.

Ist der Ausnahmefall durch einen Fehler entstanden, wird der Abbruch der Auftragsumwandlung eingeleitet.

1. eine Fehlermeldung mit Auftrags-Referenz wird ins Logbuch geschrieben
2. das HTTP-Backend wird über den Fehler informiert (Aussendung des Fehlerdokuments)
3. die Umwandlung des Auftrages in der Java-Komponente wird gestoppt

4. das HTTP-Backend erkennt einen Fehler, speichert deshalb keine Erfolgsbestätigung der Umwandlung in die Datenbank
5. das HTTP-Backend stoppt den automatischen Auftragsverarbeitungs-Prozess

6.5.2 Logging

Die Umwandlungs-Komponente schreibt Lognachrichten in eine Datei unter dem für Keksobot definierten Logverzeichnis. Das Logverzeichnis wird in der Konfigurationsdatei `Keksobot.properties` angegeben (siehe Unterabschnitt 10.4.1). Die Datei beinhaltet die Statusmeldungen, Zwischenergebnisse und Fehlermeldungen der Umwandlungs-Komponente und trägt zum Beispiel den Dateinamen `kbot-orderConverter.0.log`, wobei die Zahl die laufende Nummer darstellt. Bei jedem Start der Umwandlungs-Komponente wird eine neue Datei eröffnet (die Ziffer `0` trägt die aktuellste Datei!).

```
Mar 16, 2017 4:03:47 PM svg.objectconverters.CircleConverter convertSVGObjectToKeksobot [ordernr:
    10]
DETAIL: Converting <circle> element...

Mar 16, 2017 4:03:47 PM svg.objectconverters.CircleConverter convertSVGObjectToKeksobot [ordernr:
    10]
DETAIL: Converted <circle> element
```

Listing 6.9: Eine beispielhafte Lognachricht

Wobei aus der Lognachricht die Information auslesbar ist:

- Mar 16, 2017 4:03:47 PM ... der Zeitstempel der Lognachricht
- svg.objectconverters.CircleConverter convertSVGObjectToKeksobot ... von welcher Java-Klasse und welcher Methode die Nachricht ausgesendet wurde
- [ordernr: 10] ... die Auftragsnummer, für die dieser Programmteil aufgerufen wurde
- **DETAIL** ... es handelt sich um die Information über einen positiven Zustand (oder ein Zwischenergebnis einer Berechnung) mit geringer Wichtigkeit
- nach der Wichtigkeitsstufe folgt der Nachrichteninhalt, also *Converted <circle> element*

Kapitel 7

Vom roboterneutralen Format zur Verzierung (MH)

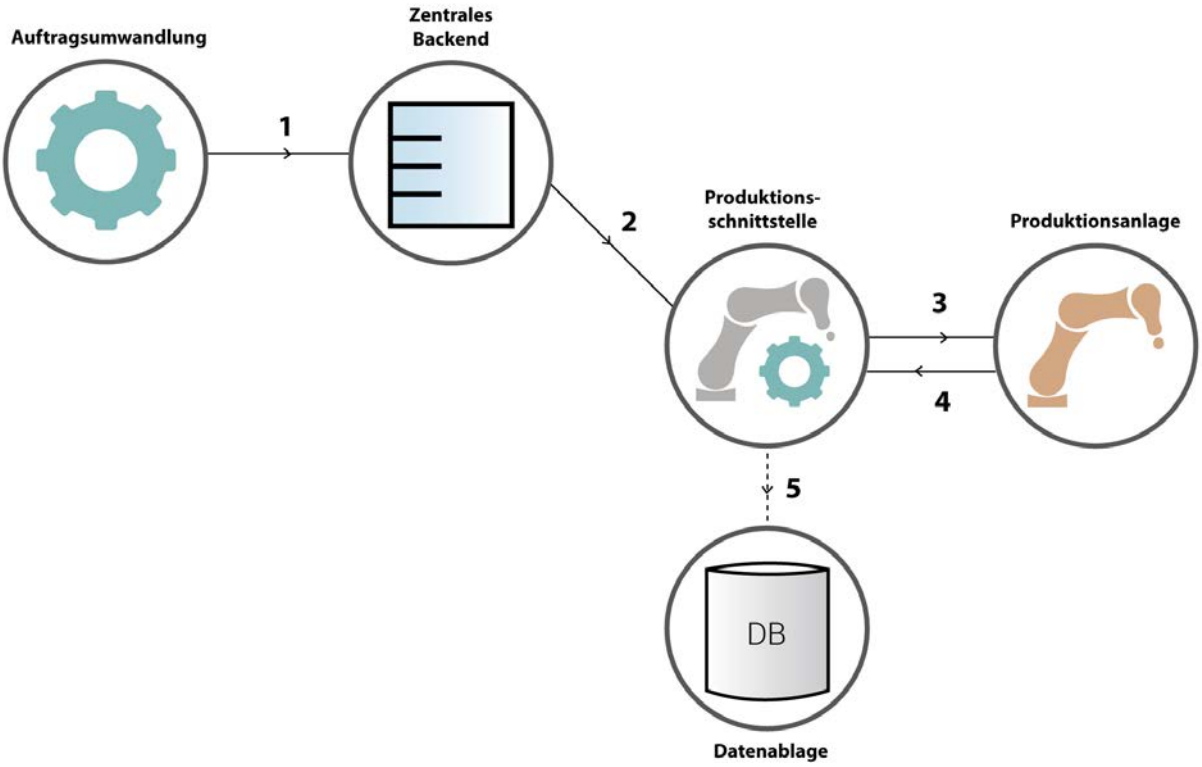


Abbildung 7.1: Übersicht des Aufbaus

7.1 Einlesen und aufbereiten des roboterneutralen Formats

7.1.1 Vorbereitung

Anhand eines einfachen Testfalls soll sichergestellt werden, dass alle Teile richtig funktionieren. In diesem beispielhaften XML, das sich an dem erstellen XML-Schema orientiert, werden alle möglichen Bewegungsarten abgebildet. Ebenfalls wird daran ersichtlich wie ein konkreter Input aussehen könnte.

```
1 <?xml version="1.0" encoding="UTF-8"?>
  <Order>
3     <ordernr>1</ordernr>
      <cookietypeid>2</cookietypeid>
```

```
    <amount>5</amount>
    <Shape>
        <metadata>
            <materialid>3</materialid>
            <velocity>4</velocity>
            <pressure>5</pressure>
        </metadata>
        <movement type="line">
            <xstart>5</xstart>
            <ystart>5</ystart>
            <xend>10</xend>
            <yend>10</yend>
        </movement>
        <movement type="curve">
            <xstart>3</xstart>
            <ystart>3</ystart>
            <xend>9</xend>
            <yend>9</yend>
            <xcurvemiddle>6</xcurvemiddle>
            <ycurvemiddle>6</ycurvemiddle>
        </movement>
        <movement type="point">
            <xstart>4</xstart>
            <ystart>4</ystart>
        </movement>
    </Shape>
    <Shape>
        <metadata>
            <materialid>3</materialid>
            <velocity>4</velocity>
            <pressure>5</pressure>
        </metadata>
        <movement type="line">
            <xstart>5</xstart>
            <ystart>5</ystart>
            <xend>10</xend>
            <yend>10</yend>
        </movement>
    </Shape>

</Order>
```

Listing 7.1: "XML Testfile"

Aufgrund der benötigten neuen Datentypen liegt es nahe eine objektorientierte Programmiersprache zu verwenden. Bei Java ist bereits Erfahrung im Umgang mit der DOM-Implementierung vorhanden, auch die vorhandene Dokumentation dazu ist sehr gut. Außerdem bietet Java den Vorteil, dass die Kompatibilität zu den folgenden Modulen, die die JOpenShowVar Java-Library verwenden, erhöht wird. Deswegen wird auch hier Java verwendet.

Die Logik befindet sich am Server auf dem sich die Backendlogik befindet, kann aber auch getrennt von diesem auf einer anderen Maschine lokalisiert sein.

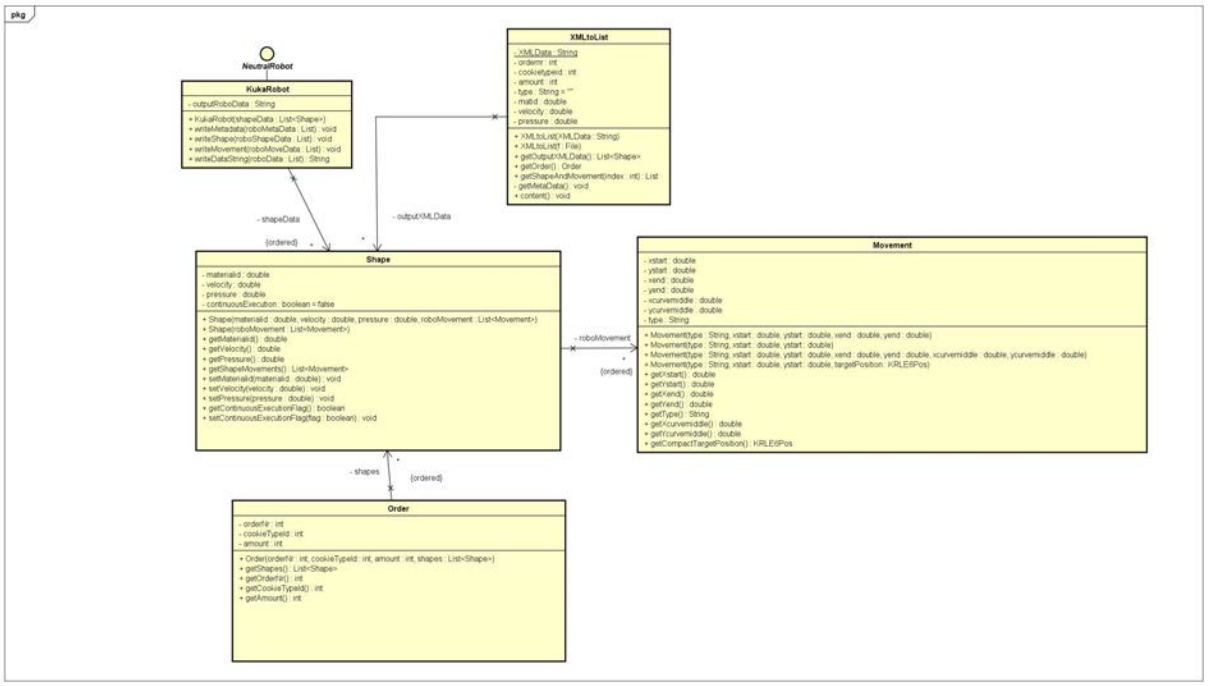


Abbildung 7.2: UML Diagramm des spezifischen Preprocessors

7.1.2 Klassenmodell

Im Folgenden werden alle Klassen der Implementierung (Siehe Abbildung 7.2) genauer beschrieben. Insgesamt gibt es drei komplexe Datentypen und eine Klasse in der der Algorithmus zum Auslesen aus dem XML implementiert wird. Außerdem gibt es ein Interface das allgemeine Robotermethoden vorgibt und eine konkrete Implementierung für den verwendeten Roboter.

Klasse Movement

Dieser Typ beschreibt eine Bewegung. Er ist ein vergleichbar umfangreicher Typ weil jede Bewegung durch ihn abgebildet wird. Deswegen verfügt er auch über mehrere Konstruktoren, sodass eine Kurve genau angegeben werden kann, aber deswegen eine linear Bewegung keine redundante Information beinhalten muss. Außerdem definiert sich ein Movement über die benötigte jeweilige Start und Endposition.
Hier wird sich konkret gegen die Verwendung von Polymorphie entschieden, da die Unterschiede in den verschiedenen Bewegungen so klein, aber die gleichen dinge so umfangreich sind, dass mehrere Unterklasse keinen Vorteil bringen würden. Sie unterscheiden sich lediglich über die Menge der Angaben in einem Bereich von 2 (point), 4 (linear Bewegung) und 5 (Halbkreis).

Klasse Shape

Shapes beinhalten mehrere Movements und Angaben zum Material, das für diese Form verwendet werden soll. So besteht eine Sternform wie in Abbildung 7.3 zu sehen aus 5 Linearen Bewegungen, wobei für alle das selbe Material verwendet wird. Sollen für eine Form unterschiedliche Materialien verwendet werden, so muss diese in mehrere Shapes unterteilt werden. Nach dem Abfahren einer Shape durch den Roboter wird jedenfalls eine Reinigungsfahrt durchgeführt. Das verhindert bei etwaigen Materialwechseln eine Verunreinigung der Verzierung.

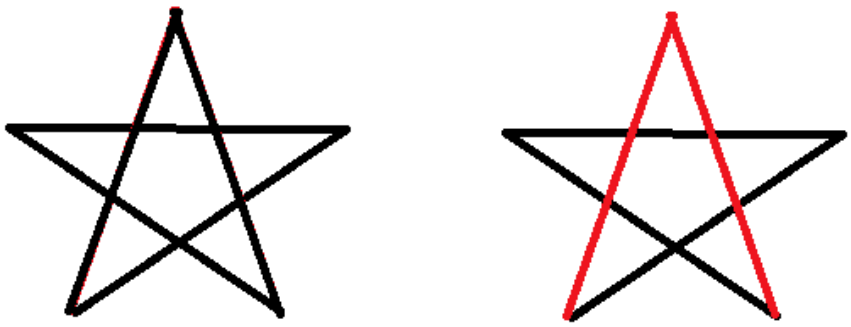


Abbildung 7.3: links: Stern als eine Shape, rechts: Stern in zwei Shapes

Klasse Order

Die Klasse Order behandelt Bestellungen, die alle aufzurufenden Movements und alle notwendigen Metadaten wie Nummer, Menge und Kekstyp, um die Bestellung ausführen zu können, beinhaltet. Der aufrufende Daemon nimmt nur Objekte des Typs "Order" entgegen, da er nicht mit Shapes alleine arbeiten kann.

Klasse XML to List

Die Aufgabe der XMLtoList Klasse besteht darin, die Informationen aus dem roboterneutralen XML-Format herauszufiltern um sie für die Weiterverarbeitung und den Transfer an den Roboter vorzubereiten. Als Input wird sowohl eine Datei als auch ein String akzeptiert. Es wird entweder eine Liste mit Shapes, die sogenannte "OutputXMLData", oder ein neues Order-Objekt, mit den Daten der momentan bearbeiteten Bestellung, zurück gegeben. Eine Übersicht über den Ablauf kann im Ablaufdiagramm (Siehe Abbildung 7.4) gefunden werden.
Die Verarbeitung der Daten erfolgt mittels "DOM (Doucment Object Model)". Der Zugriff auf das XML Dokument wird hier über einen DOM-Baum realisiert. Das ist eine hierarchisch Struktur die sich in sogenannte "Nodes", also Knoten, unterteilt. Ein Knoten kann mehrere Kind-Elemente haben. Außerdem haben alle Elemente, abgesehen vom "Wurzelelement", "Eltern" (ihre darüberliegenden Nodes). Das Wurzelelement ist das höchste Element im DOM Baum. Wird mit dem DOM Baum gearbeitet so ist das Wurzelelement meist auch der Ausgangspunkt für das Programm, da alle Kindelemente von ihm aus erreichbar sind.

```
1 private void getMetaData(){
    ordernr = Integer.parseInt(child.get(0).getTextContent());
3   cookietypeid = Integer.parseInt(child.get(1).getTextContent());
    amount = Integer.parseInt(child.get(2).getTextContent());
5 }
```

Listing 7.2: Auslesen der Metadaten

Metadaten, wie die Nummer der Bestelung oder der verwendete Keks, werden direkt im Konstruktor heraus gefiltert. Allerdings bezieht sich das Programm hier auf die genaue Position der Einträge. Wie in dem Codebeispiel (Siehe Listing 7.2) beschrieben. Das ist möglich weil in dem XML Schema festgelegt wurde welche Elemente in welcher Reihenfolge vorkommen dürfen. Die Validierung des XMLs findet

bereits statt bevor es zum Preprocessor kommt. So ist es gar nicht möglich hier Fehlschläge zu verursachen.

```
for(int i = 2; i < child.size() ; i=i+1){
  Node docChil = child.get(i);
  LOGGER.log(Level.FINEST, "Processing child of root: "+docChil.getNodeName());

  if (docChil.getNodeName().equals( "Shape")){
      //System.out.println(i);
      List<Movement> ShapeMovements = getShapeAndMovement(i);
      for(int j = 0; j < ShapeMovements.size(); j++){
          //System.out.println(ShapeMovements.get(j).getType());
      }
      Shape s = new Shape(matid,velocity,pressure,ShapeMovements);
      outputXMLData.add(s);
```

Listing 7.3: Auszug Methode content()

Die restlichen Informationen werden in der Methode content() (Siehe Listing 7.3) ausgelesen. Hier werden alle Shapes aus dem Baum extrahiert und in einer Liste zwischengespeichert. Zu jeder Shape wird die Methode getShapeAndMovement(), mit dem entsprechenden index (= Schritte in den DOM Baum hinein) aufgerufen, um die Movements aus dieser Shape zu extrahieren. Die neue Shape wird an die Liste OutputXMLData angehängt.
Sollte hier ein Fehler auftreten wirft das Programm eine "KbotApplicationException" die den aufgetretenen Fehler in den eigenen Log-Files einträgt.

```
Node shap = shapes.item(i);

 if(!(shap instanceof Element)) {
     continue;
 }

 if(shap.getNodeName().equals("metadata")){
         //Liste mit metadaten
         NodeList nmet = shapes.item(i).getChildNodes();
         for(int j = 0; j < nmet.getLength(); j=j+1) {

                 //Elemente von Movement
                 Node met = nmet.item(j);

                 if(!(met instanceof Element)) {
                     continue;
                 }

                 if (met.getNodeName().equals("materialid")) {
                         matid = Integer.parseInt(met.getTextContent());
                 } else if (met.getNodeName().equals("velocity")) {
                         velocity = Double.parseDouble(met.getTextContent());
                 } else if (met.getNodeName().equals("pressure")) {
                         pressure = Double.parseDouble(met.getTextContent());
                 }
         }
 }
```

Listing 7.4: ShapesAndMovement Auszug

In der Methode getShapeAndMovements() (Siehe Listing 7.4) wird zunächst sicherheitshalber überprüft ob es sich bei dem Objekt auch um ein Element handelt, da der Vorgang ansonsten fehlschlagen wird. Danach müssen die Metadaten der Shape ausgelesen werden.

```
//movements
if(shap.getNodeName().equals("movement")) {

    //Attribut
    NamedNodeMap movementAttr = shapes.item(i).getAttributes();
    for(int at = 0; at < movementAttr.getLength(); at++){
        Node attr = movementAttr.item(at);
        type = attr.getTextContent();
    }
    //Liste mit movementdaten
    NodeList nmov = shapes.item(i).getChildNodes();
```

Listing 7.5: Filtern der Attribute aus einem Movement

Handelt es sich nun um ein Movement so werden vorerst die Attribute gefiltert und danach eine Liste mit den einzelnen Elementen des Movements erstellt wie im Listing Listing 7.5 zu sehen ist.

```
for(int j = 0; j < nmov.getLength(); j=j+1) {

    //Elemente von Movement
    Node mov = nmov.item(j);

    if(!(mov instanceof Element)) {
      continue;
    }

    if(mov.getNodeName().equals("xstart")){
        xstart = Double.parseDouble(mov.getTextContent());
    }else if(mov.getNodeName().equals("ystart")) {
        ystart = Double.parseDouble(mov.getTextContent());
    }else if(mov.getNodeName().equals("xend")){
        xend = Double.parseDouble(mov.getTextContent());
    }else if(mov.getNodeName().equals("yend")){
        yend = Double.parseDouble(mov.getTextContent());
    }else if(mov.getNodeName().equals("xcurvemiddle")){
        xcurvemiddle = Double.parseDouble(mov.getTextContent());
    }else if(mov.getNodeName().equals("ycurvemiddle")){
        ycurvemiddle = Double.parseDouble(mov.getTextContent());
    }
    }
    if(type.equals("point")){
    ShapesAndMovements.add(new Movement(type,xstart,ystart));
    }else if(type.equals("circularArc")){
    ShapesAndMovements.add(new Movement(type,xstart,ystart,xend,yend,xcurvemiddle,ycurvemiddle));
    } else {
    ShapesAndMovements.add(new Movement(type,xstart,ystart,xend,yend));
    }
```

Listing 7.6: Filtern der Daten aus einem Movement

Diese Liste wird danach wieder durchgegangen, um den Inhalt der einzelnen Elemente zu erhalten, wobei hier auf den Typ der Bewegung geachtet wird. Folgende Attribute müssen vorhanden sein :

- xstart
- ystart
- xend

- yend

Andere Attribute wie xcurvemiddel und ycurvemiddle werden nur für Kurven benötigt. Danach wird je nach Art der Bewegung ein anderer Konstruktor der Movement Klasse verwendet um das passende Objekt zu erstellen. Das passiert in einer Anhäufung von if-else statements wie im Listing 7.6 zu sehen ist.

Interface NeutralRobot

Hier handelt es sich um ein Interface mit Methoden die jede Roboterimplementierung haben muss. (Siehe Listing 7.9) Diese dienen im aktuellen Anwendungsfall zu nichts, weil die JOpenShowVar-Library sie überflüssig macht. Wird allerdings nicht mit einem KUKA-Roboter gearbeitet sondern mit einem Roboter der fertige Befehle empfangen kann, so werden diese Methoden implementiert und dort die gegebenen Daten in direkte Bewegungsbefehle übersetzt. Beispielsweise würde sich dieser Input aus dem XML:

```
<movement type="line">
    <xstart>5</xstart>
    <ystart>5</ystart>
    <xend>10</xend>
    <yend>10</yend>
</movement>
```

Listing 7.7: exemplarische Bewegung in XML

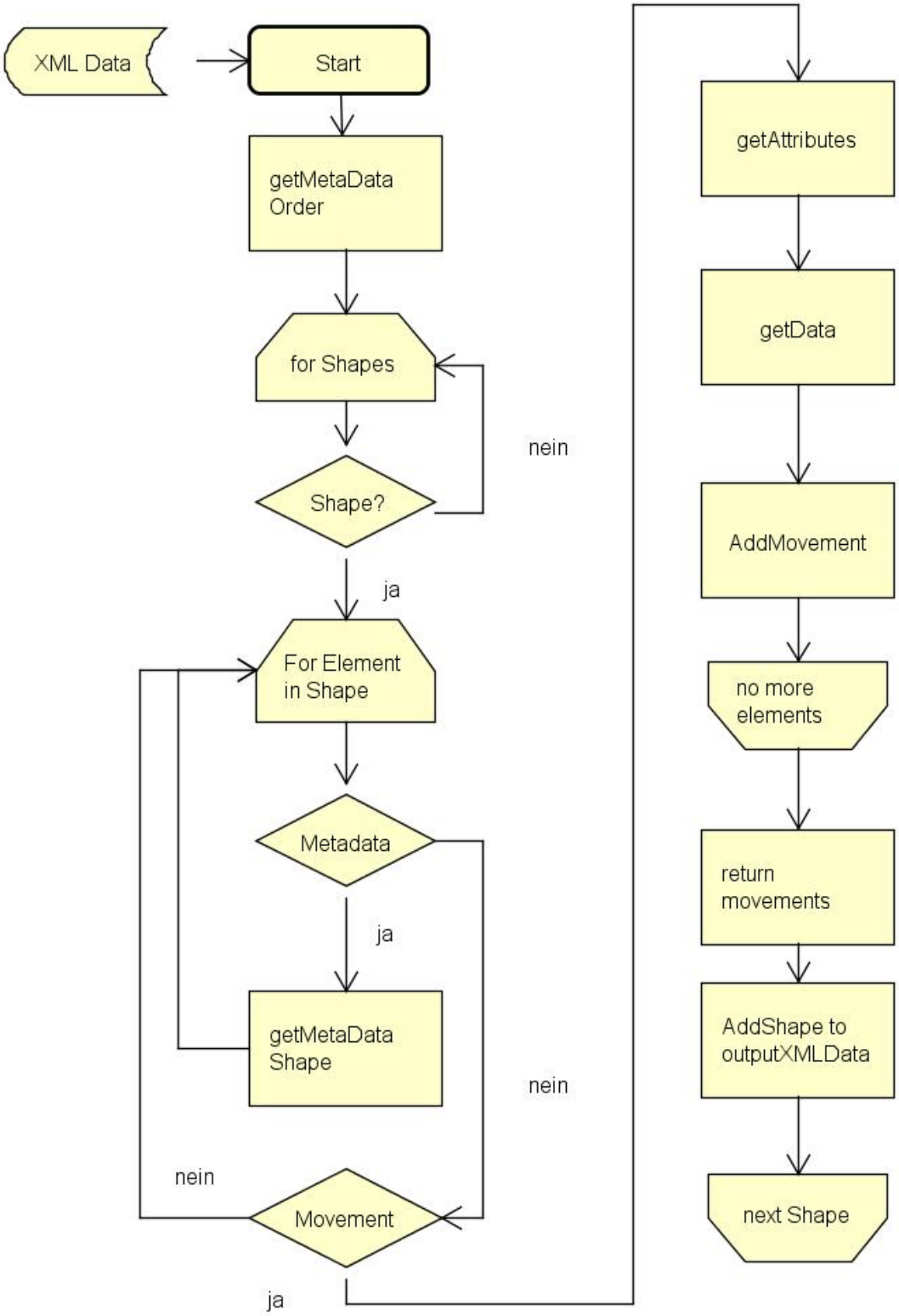


Abbildung 7.4: Ablaufdiagramm XMLtoList

In folgenden KRL (KUKA Robot Language) -Bewegungsbefehl übersetzen:

```
...
;zuvor wird der punkt ziel mit den gegegbenen variablen befüllt
LIN ziel
```

Listing 7.8: exemplarische Bewegungsübersetzung in KRL

Im Anschluss könnte dieser fertige Befehl, sofern es der Roboter zulässt, direkt übertragen werden.

```
//takes metadata and writes it in the output String
 public void writeMetadata(List<Shape> roboMetaData);
 //takes shapes and writes it in the output String, uses writeMovement
 public void writeShape(List<Shape> roboShapeData);
 //takes movements and writes it in the outputString, used by writeShape
 public void writeMovement(List<Shape> roboMoveData);
 //takes robo data as a list and writes it in the output String, uses all of the above methods
 public String writeDataString(List<Shape> roboData);
```

Listing 7.9: Methoden des Interfaces

Klasse KukaRobot

KukaRobot ist nun eine konkrete Implementierung des NeutralRobot Interfaces. Da eine Übersetzung auf "KRL (Kuka Robot Language)" nicht nötig ist bleibt diese Implementierung zwar formal vorhanden, wird im weiteren Verlauf allerdings nicht benötigt. In Zukunft könnte man ein System entwickeln das das direkte Senden der Befehle an den Roboter ermöglicht, dann würde diese Implementierung genutzt werden. Durch diese Art der Übertragung würde der Roboter immens entlastet werden, da ihm das "Hören" auf einen Socket und das Hantieren mit den Variablen erübrigt bleibt.

7.2 Verbindung zum Roboter

Die Verwendung von JOpenShowVar funktioniert über das Netzwerk. Beide Parteien, also die Produktsanlage und das zentrale Backend auf dem sich die Produktionsschnittstelle befindet, müssen sich in diesem befinden. Dann werden über die Library Daten, für die globalen Variablen am Roboter, an einen Socket gesendet. Am Roboter läuft ein Programm das den Socket ausliest und die Daten an das Programm vom KUKA-Roboter überträgt.(Siehe Abbildung 7.5) Der verwendete Port ist 7000. Ein belegter Port würde zum sofortigen Abbruch des Programms führen. Deswegen wird auch ein so "ausgefallener", also selten verwendeter Port, genutzt. Ein standadisierter und deswegen nicht ratsamer Port wäre beispielsweise **8080**, der standard HTTP Port. Er wird meistens von Serverapplikationen belegt und ist daher keine gute Wahl.

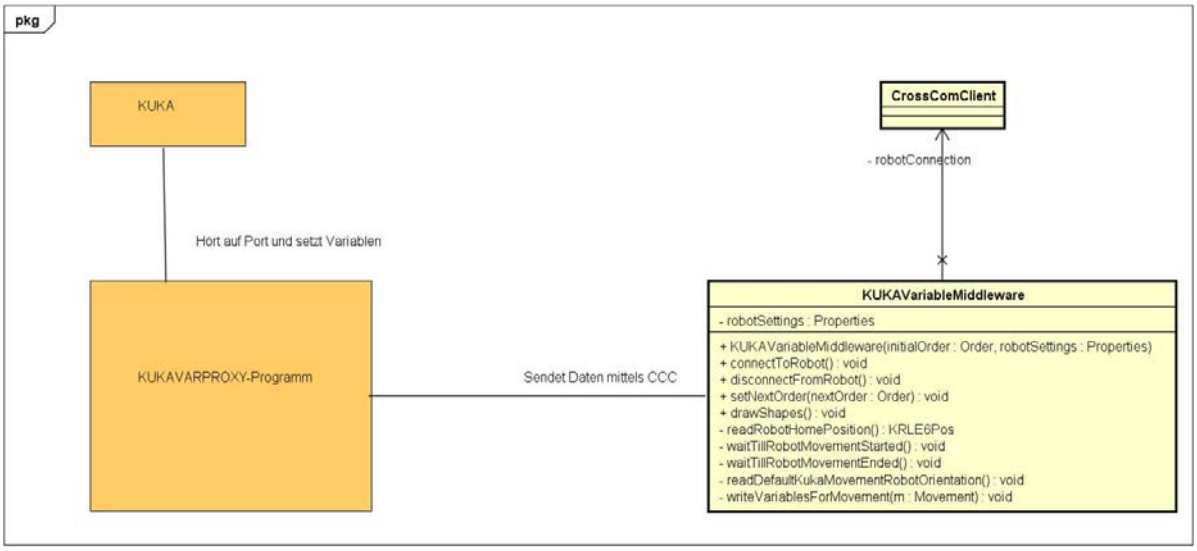


Abbildung 7.5: Darstellung der Kommunikation zwischen dem Java Programm und dem Roboter

7.2.1 Voraussetzungen an die Produktionsebene

Globale Variablen zur Kommunikation

Globale Variablen lassen sich in einer config-Datei des Roboters setzen. (Siehe Unterabschnitt 10.2.2) Diese Variablen (Siehe Listing 7.10) müssen global sein damit auch das am Roboter befindliche Programm, das die Daten entgegen nimmt, auf sie zugreifen kann. Sie sind vorgesehen, um die möglichen Bewegungssatz-Typen und ihre Parameter abzudecken.

```
BOOL KBOT_READY_FOR_MVMT
BOOL KBOT_MVMT_STARTSIGNAL
INT KBOT_MVMT_TYPE
E6POS KBOT_START_POSITION
E6POS KBOT_TARGET_POSITION
REAL KBOT_VELOCITY
E6POS KBOT_SUPPORTING_POSITION
```

Listing 7.10: benötigt Variablen am Roboter

Mit `KBOT_READY_FOR_MVMT` signalisiert der Roboter der Produktionsschnittstelle, dass gerade keine Bewegung stattfindet und er deshalb bereit für den nächsten Bewegungssatz ist. Wenn der übergeordnete Controller dem Roboter die Erlaubnis für den Start der Bewegung gibt (via des `KBOT_MVMT_STARTSIGNAL`), werden die Variablen kopiert und danach wird die `KBOT_READY_FOR_MVMT` auf FALSE gesetzt. Das soll dem Controller signalisieren, dass der Startschuss erhalten wurde und die zugehörige Bewegung ausgeführt wird.
Der Roboter verwendet für die tatsächliche Ausführung nicht die `KBOT_xxxxxxxxx` Variablen, sondern seine eigene Kopie der Variablen.
Der Vorteil davon ist, dass der übergeordnete Controller den Start der Roboterbewegung bemerkt (er fragt zyklisch die Variable `KBOT_READY_FOR_MVMT` ab) und währenddessen schon die Parameter für die nächste Bewegung an den Roboter überträgt.

Die Variable `KBOT_MVMT_STARTSIGNAL` wird von der Produktionsschnittstelle auf TRUE gesetzt, nach-

dem die Variablen für den nächsten Bewegungssatz gesetzt wurden. Erst nach diesem Startschuss wird der Bewegungsbefehl am Roboter ausgeführt. Ansonsten würde sich der Roboter sofort, vielleicht mit veralteten Variablenwerten von der vorigen Bewegung, weiterbewegen.

Die Variable `KBOT_MVMT_TYPE` entscheidet darüber, welche Bewegungsart gefahren wird. Mögliche Werte für `KBOT_MVMT_TYPE` sind:

Wert	**Bedeutung des Werts für den Roboter**	**Benötigte KBOT-Variablen**
KBOT_MVMTTYPE_LIN	Fahre Linearbewegung	`KBOT_TARGET_POSITION`
KBOT_MVMTTYPE_CIRC	Fahre Zirkularbewegung	`KBOT_TARGET_POSITION`, `KBOT_SUPPORTING_POSITION`
KBOT_MVMTTYPE_PTP	Fahre PTP-Bewegung	`KBOT_TARGET_POSITION`
KBOT_MVMTTYPE_START	Keksverzierung wird begonnen, fahre erste definierte Position der Verzierung an	`KBOT_TARGET_POSITION`
KBOT_MVMTTYPE_END	Keks fertig verziert, fahre zur HOME-Position	keine

Der Punkt `KBOT_SUPPORTING_POSITION` dient dazu, Kreisbögen eindeutig zu definieren. Eine *CIRC*-Bewegung benötigt diesen Extraparameter.

Freigabe des Kommunikationskanals

Das KUKAVARPROXY.exe ist ein Programm das mit der JOpenShowVar-Library mitgeliefert wird und am Roboter läuft. Wie bereits erwähnt sendet das Java Programm die Daten auf einen bestimmten Port des Roboters. Das Programm diet nun dazu diese Daten aus dem Port auszulesen und die entsprechenden Variablen im Roboterprogramm zu setzen.
Damit das Programm funktionieren kann muss der enstprechende Port, in diesem Fall der Port 7000, freigegeben werden. Wäre er geschlossen so könnte das Java Programm sich nicht darauf verbinden und würde eine Fehlermeldung zurück geben. Auch der Roboter würde nie an seine Variablen kommen. Dieses Programm "lauscht" aktiv auf den zuvor freigegebenen Port 7000 und empfängt die Werte für die Variablen.

Das Listing 7.11 zeigt eine KRL-Umsetzung die mithilfe von dynamisch gesetzten Variablen ein Roboterprogramm steuert und wie diese genutzt werden können. Alle KBOT_xxxxx Variablen werden vor dem Beginn des Programmes von der Produktionsschnittstelle gesetzt. So können sie statisch verwendet werden aber trotzdem dynamische Werte beinhalten. Das Beispiel zeigt wie anhand des Bewegungstypen (KBOT_MVMT_TYPE) ein konkreter Bewegungsbefehl am Roboter gewählt wird.

```
1 DEF Modul( )
  INI
3 PTP HOME VEL=100 % DEFAULT

5 IF (KBOT_MVMT_TYPE==0) THEN
    LIN KBOT_TARGET_POSITION VEL=KBOT_VELOCITY
7
  ELSE IF (KBOT_MVMT_TYPE==1) THEN
```

```
  CIRC KBOT_SUPPORTING_POSITION KBOT_TARGET_POSITION VEL=KBOT_VELOCITY

ELSE IF (KBOT_MVMT_TYPE==2) THEN
  PTP KBOT_TARGET_POSITION VEL=100
ENDIF

PTP HOME VEL=100 % DEFAULT
END
```

Listing 7.11: Beispielhaftes dynamisches KRL Programm

Alle Informationen zu den notwendigen Schritten um das Programm zu starten und den Port freizugeben finden sich bei der Erstbetriebnahme Erklärung im Unterabschnitt 10.2.1.

Konfiguration der Betriebslampen um den Status der Verzierung anzuzeigen

In der Testumgebung stehen Leuchten in einem Ampelgehäuse zur Verfügung. Sie sollen in der KRL-Komponente dazu verwendet werden, einem Bediener einen Status des Verzierungsprozesses anzuzeigen. Wenn der Keks fertig verziert ist und der Roboter in der Ausgangsposition steht, dann leuchtet die grüne Betriebslampe. Während des Vorgangs der Verzierung leuchtet die gelbe Betriebslampe. Die rote zeigt eine Not-Halt Situation oder andere ungeplante Unterbrechungen des Verzierungsvorganges an.

Die Betriebslampe ist mit einem digiteln Output-Modul (Wago 750), das über den KUKA-Roboter Extensionbus via EtherCat von der Robotersteuerung geschalten werden kann, verbunden.(Siehe Abschnitt Unterabschnitt 10.2.3)

Die Ausgänge sind wie in Abschnitt 7.2.1 belegt.

- $OUT[21] für die grüne “Work done” Betriebslampe
- $OUT[22] für die gelbe “Work in Progress” Betriebslampe
- $OUT[23] für die rote “Fehler” Betriebslampe

In KRL-Programmen können sie dann zum Beispiel wie in Listing 7.12 angesprochen werden:

```
$OUT[21] = TRUE ; turn on 'work done' light
```

Listing 7.12: Ansprechen einer Betriebslampe

7.2.2 Properties-File für das Java Programm

Um eine Verbindung zur Übertragung der Daten herzustellen muss auch die Java Applikation entsprechend eingerichtet werden. Die notwendigen Daten wie Hostname (IP-Adresse) und Port werden aus einem Properties-File (Siehe Unterabschnitt 10.4.1) ausgelesen indem diese vermerkt sind. Außerdem sind hier auch Defaultwerte für die Achsausrichtung und die Homeposition hinterlegt. Sollten diese Daten aus irgendeinem Grund fehlerhaft oder nicht vorhanden sein so wird das Programm mit einer Fehlermeldung abgebrochen.

7.3 Senden an den Roboter

7.3.1 Java Umsetzung

Die Lösung setzt auf die Bibliothek *JOpenShowVar*[1]. Ein dynamisches Roboterprogramm wird erreicht, indem dem Roboter globale Variablen von einer externen Einheit, in diesem Fall das Java-Programm, neu definiert und dann im Roboterprogramm verwendet werden. Dabei stellt die oben erwähnte Softwarebibliothek JOpenShowVar die Brücke zwischen Variablen-setzender Produktionsschnittstelle, auf dem zentralen Backend, und dem Roboterprogramm, auf der Robotersteuerung, her.

Das Roboterprogramm entscheidet sich also je nach vom Java-Programm gesetzen Variablenwert für einen Bewegungsbefehl. Die wichtigste Methode des zugehörigen Java-Programms hat folgenden Quellcode:

```
1  public void drawShapes() throws IOException {
       /* copy Order object of active order because it can be overwritten */
3      List<Shape> orderShapes = new LinkedList<Shape>(this.activeOrder.getShapes());

5    /* robot control variables */
     KRLBool robot_movementStartsignal = new KRLBool("KBOT_MVMT_STARTSIGNAL");
7    KRLBool robot_readyForNextMovement = new KRLBool("KBOT_READY_FOR_MVMT");

9    /* robot movement attributes */
     KRLReal robot_velocity = new KRLReal("KBOT_VELOCITY");
11
     /* give robot control variables an initialization value */
13   robot_movementStartsignal.setValue(false);
```

Im ersten Schritt werden wird eine neue Liste mit den zu verzierenden Shapes geholt, controlling Variablen erstellt und initialisiert.

```
1    for(Shape s : orderShapes) {

3      /* wait until robot signals it is ready */
       this.robotConnection.readVariable(robot_readyForNextMovement);
5      while(robot_readyForNextMovement.getValue() == Boolean.FALSE) {

7        this.robotConnection.readVariable(robot_readyForNextMovement);
         try {
9          Thread.sleep(60);
         } catch (InterruptedException e) {
11           LOGGER.log(Level.WARNING, "Something tried to interrupt!");
         }
13     }

15     /* set variables which are always the same for all movements of the current shape */
       robot_velocity.setValue(s.getVelocity());
17     this.robotConnection.writeVariable(robot_velocity);

19     /*
        * Prepares the robot for the first movement.
21      * the first movement is special (the order of commands is different),
        * therefore it is located outside the main movement loop.
23      */
       Movement firstMovement = s.getShapeMovements().get(0);
25     writeVariablesForMovement(firstMovement);

27     robot_movementStartsignal.setValue(true);
```

```
this.robotConnection.writeVariable(robot_movementStartsignal);
```

Listing 7.13: Shape Iteration

Dann Beginnt die Iteration durch die erwähnten Shapes. Wenn der Roboter bereit für die nächste Shape ist werden die Variablenwerte und das Startsignal auf TRUE gesetzt.

```
1
   /* iterate over the movements of the current shape */
3  List<Movement> allMovements = s.getShapeMovements();
   List<Movement> otherMovements = allMovements.subList(1, allMovements.size());
5  for(Movement m : otherMovements) {

7    /* first we have to wait for the previous movement to begin */
     this.waitTillRobotMovementStarted();
9    // set the movement startsignal back to false since we now prepare a new movement which the
         robot has no permission yet
     robot_movementStartsignal.setValue(false);
11   this.robotConnection.writeVariable(robot_movementStartsignal);


13
     /* then we can set the variables for the next movement while the robot executes it */
15   writeVariablesForMovement(m);

17   /* we wait till the roboter signals that it finished the previous movement */
     this.waitTillRobotMovementEnded();
19
     /* the previous movement has ended, now we give the robot permission to execute this movement
         */
21   robot_movementStartsignal.setValue(true);
     this.robotConnection.writeVariable(robot_movementStartsignal);
23
   } // end of movements iteration
25

   } // end of shapes iteration
```

Listing 7.14: Movement Iteration

Danach wird durch die Movements der aktuellen Shape iteriert. Nachdem das aktuelle Movement gestartet wurde wird das Startsignal wieder auf FALSE beginnt, damit, falls der Roboter schneller ist als das Programm, die selbst Fahrt nicht nocheinmal durchgeführt wird. Ist er fertig mit seiner Bewegung wird das Startsignal wieder auf TRUE gesetzt. Nachdem der Roboter die Letzte Bewegung einer Shape begonnen hat beginnt der Reinigungsvorgang. Mehr zum Sinn der Reinigung und der konkreten Implementierung kann in Abschnitt 9.3 gefunden werden.

```
1
  /**
3  * Signal robot to move to HOME position
   */
5 /* wait until the robot started the last actual shape movement */
  waitTillRobotMovementStarted();
7 // set the movement startsignal back to false since we now prepare a new movement which the robot
      has no permission yet
  robot_movementStartsignal.setValue(false);
9 this.robotConnection.writeVariable(robot_movementStartsignal);
```

Listing 7.15: Robotervorbereitung für HOME Anweisung

Nach Beginn der letzten Bewegung der letzte Shape werden die Variablen für die Fahrt zur HOME-Position gesetzt.

```
2   /* initiate the end movement to HOME */
4   // find end position of last shape -> will be start position for HOME movement
    List<Shape> activeOrderShapes = this.activeOrder.getShapes();
6   Shape lastShape = activeOrderShapes.get(activeOrderShapes.size()-1);
    List<Movement> lastShape_movements = lastShape.getShapeMovements();
8   Movement lastShape_lastMovement = lastShape_movements.get(lastShape_movements.size()-1);
10  // read HOME position from robot
    KRLE6Pos homePosition = this.readRobotHomePosition();
12
    // build Kbot Movement object and execute it on robot
14  Movement mvmtToHome = new Movement("endmovement", lastShape_lastMovement.getXend(),
        lastShape_lastMovement.getYend(), homePosition);
16  // write Kbot Movement to end position to robot
    writeVariablesForMovement(mvmtToHome); position.");
18
    // wait till the KUKA robot finished the previous movement
20  this.waitTillRobotMovementEnded();
```

Listing 7.16: Roboter fahre zu HOME Anweisung

Die letzte Bewegung ist die Fahrt zu HOME.

```
1
    // give startsignal for end movement on robot
3   robot_movementStartsignal.setValue(true);
    this.robotConnection.writeVariable(robot_movementStartsignal);
5   this.robotConnection.readVariable(robot_movementStartsignal);
7
    /*
9    * Set startsignal back to false so that the robot waits until the next order is processed
     * Otherwise the last movement would be repeated over and over again until the next order is
11   * processed.
     */
13  waitTillRobotMovementStarted();
    robot_movementStartsignal.setValue(false);
15  this.robotConnection.writeVariable(robot_movementStartsignal);

17 }
```

Listing 7.17: Nachbereitung

Danach wird auf neue Arbeitsaufträge gewartet.

Wie gesehen werden kann ist wird also durch alle Shapes iteriert, und für jede Shape nochmal durch alle Movements die sie beinhaltet. Für jedes Movement wird gecheckt ob er Roboter bereits den Verziervorgang gestartet hat, wenn ja wird das Startsignal auf False gesetzt und die Variablen mit den entsprechenden Werte (xstart, ystart, xend, yend, und andere) befüllt. Nach dem letzten Movement einer Shape reinigt sich der Roboter automatisch indem er die dafür definierten Punkte anfährt und die Drehbewegung durchführt. Nach dem alle Shapes durchgegangen wurden wird er zu seiner “Home Position” zurück

geschickt.
Besondere Beachtung wird hier der ersten Bewegung geschenkt, da hier die Abfolge and Befehlen anders sein muss. Der Roboter hat nämlich noch keinen Ausgangspunkt in der Form sondern startet von seiner Home Position. Daher wird er hier eine "PTP-Bewegung(Point-to-Point, immer die schnellstmögliche Bewegung, nicht zwingend linear)" zum Startpunkt der Shape gemacht. Der Vorgang wird für die in der Bestellung angegebene Anzzahl von Keksen wiederholt.

Das Ende eines Durchlaufes wird via Betriebslampe signalisiert,sodass ein Mitarbeiter einen Kekse nachlegen kann. Nachdem dieser den Gefahrenbereich verlassen hat signalisiert er das durch einen Input, da der implementierte Prototyp nur darauf ausgelegt ist einen Keks an einer Stelle zu verzieren. Sollen automatisch mehrere Kekse verziert werden so müsste die Shape- und Movementkoordinaten jedes Mal um die Versetzung weiter gerechnet oder mit mehreren Koordinatensystemen im Roboter gearbeitet werden. In der Realität würde das Auswechseln auch nicht durch einen Mitarbeiter passieren sondern ein Förderband, das nach einem Durchgang um eine definierte Länge weiterfahrt. Danach würde der nächste, noch unverzierte Keks wieder an der korrekten Stelle für den Roboter liegen. Diese Aufgabe an ein Förderband anstatt an die Applikation zu übergeben entlastet dieses ungemein.

7.3.2 KRL Umsetzung

Der KUKA-Roboter kann Programme speichern und ausführen. Dazu wird das passende Programm einfach in der Liste der Programme angewählt. Genau das wird genutzt um die von der Java Applikation gesetzt Variablen automatisch anzufahren. Das entsprechened Programm sieht wie folgt aus:

```
  &ACCESS RVP
&REL 1
&PARAM DISKPATH = KRC:\R1\Program\Keksobot
DEF Kbot_DrawShapes_Program( )
;FOLD INI
  ;FOLD BASISTECH INI
    GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
    INTERRUPT ON 3
    BAS (#INITMOV,0 )
  ;ENDFOLD (BASISTECH INI)
  ;FOLD USER INI
    ;Make your modifications here

  ;ENDFOLD (USER INI)
;ENDFOLD (INI)

;FOLD PTP HOME  Vel= 100 % DEFAULT;%{PE}%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:HOME, 3:, 5:100, 7:
    DEFAULT
$BWDSTART = FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS (#PTP_PARAMS,100 )
$H_POS=XHOME
PTP  XHOME
;ENDFOLD

; debug variables
HANDLE = 0
MODE=#SYNC
```

Listing 7.18: Vorbereitung SAK-Fahrt

Zu Beginn macht der Roboter eine sogenannte “SAK(Satzkoinzidenz)-Faht” zu seiner HOME-Position.

```
WHILE (TRUE == TRUE)

; prepare robot to use our base and tool system
BAS(#BASE, 11)
BAS(#TOOL, 11)

; ;;;;;;;
; set robot velocity for the received movement command
; ;;;;;;;
COPY_VELOCITY=KBOT_VELOCITY
; check if requested velocity exceeds current velocity limits
IF COPY_VELOCITY >= $VEL_MA.CP THEN
$OV_PRO=100 ; counts for PTP movements
ELSE
$OV_PRO=COPY_VELOCITY/$VEL_MA.CP*100 ; counts for PTP movements
ENDIF
```

Listing 7.19: Vorbereitung der Keksverzierung

Dann Beginnt die Endlosschleife in der zuerst die Metadaten-Variablen kopiert werden.

```
; continue to wait until upper-level controller gives permission to start
WAIT FOR KBOT_MVMT_STARTSIGNAL==TRUE
```

Listing 7.20: Warten auf Startsignal

Dann muss der Roboter warten bis er das Startsignal vom Java-Programm bekommt.

```
; turn off 'work-in-progress' and 'work-done' light
$OUT[21] = FALSE
$OUT[22] = FALSE
```

Listing 7.21: Einstellen der Betriebslampen

```
; mvmt parameters are copied so that the upper-level controller can write new ones
COPY_MOVEMENT_TYPE=KBOT_MVMT_TYPE
COPY_START_POSITION=KBOT_START_POSITION
COPY_TARGET_POSITION=KBOT_TARGET_POSITION
COPY_SUPPORTING_POSITION=KBOT_SUPPORTING_POSITION
```

Listing 7.22: Kopieren der Variablen

Jetzt werden die Variablen für die konkrete Bewegung kopiert.

```
; signal upper-level controller that robot starts the movement
KBOT_READY_FOR_MVMT=FALSE

; turn on the 'work-in-progress' light
$OUT[22]=TRUE

; This fixes a bug where our upper-level controlling program does not recognize the
    KBOT_READY_FOR_MVMT change fast enough
;WAIT SEC 0.08 ; it has to be longer than the thread of the upper-level program sleeps
WAIT FOR KBOT_MVMT_STARTSIGNAL == FALSE

; at the start of every movement the robot moves to the specified START POSITION
; this is done so that the order of movements in the upper-level robot-neutral document does not
    matter
```

```
;CWRITE($FCT_CALL, STAT, MODE, "krl_fopen", "kbot_drawShapesNew.txt", "a", HANDLE)
;CWRITE($FCT_CALL, STAT, MODE, "krl_fprintf", HANDLE, "%d", ($DATE.MIN*60+$DATE.SEC))
;CWRITE($FCT_CALL, STAT, MODE, "krl_fprintf", "Problematic line will start now%s", "'HOD''HOA'")
;CWRITE($FCT_CALL, STAT, MODE, "krl_fprintf", HANDLE,"APOS,%+#10.4f,%+#10.4f,%+#10.4f,%+#10.4f
    ,%+#10.4f,%+#10.4f%s", $AXIS_ACT.A1, $AXIS_ACT.A2, $AXIS_ACT.A3, $AXIS_ACT.A4, $AXIS_ACT.A5,
    $AXIS_ACT.A6, "'HOD''HOA'")
IF COPY_MOVEMENT_TYPE <> #KBOT_MVMTTYPE_END THEN
PTP COPY_START_POSITION
ENDIF
;CWRITE($FCT_CALL, STAT, MODE, "krl_fprintf", HANDLE, "%d", ($DATE.MIN*60+$DATE.SEC))
;CWRITE($FCT_CALL, STAT, MODE, "krl_fprintf", HANDLE,"Problematic line ended%s", "'HOD''HOA'")
;CWRITE($FCT_CALL, STAT, MODE, "krl_fprintf", HANDLE,"APOS,%+#10.4f,%+#10.4f,%+#10.4f,%+#10.4f
    ,%+#10.4f,%+#10.4f%s", $AXIS_ACT.A1, $AXIS_ACT.A2, $AXIS_ACT.A3, $AXIS_ACT.A4, $AXIS_ACT.A5,
    $AXIS_ACT.A6, "'HOD''HOA'")
;CWRITE($FCT_CALL, STAT, MODE, "krl_fclose", HANDLE)
```

Listing 7.23: An Controller und Betriebslampe Start signalisieren

Der Roboter signalisiert dann, durch einschalten der "Work-in-Progress" Betriebslampe und setzen der KBOT_READY_FOR_MVMT Variable auf FALSE, dass er jetzt arbeitet und keine Daten empfangen kann. Außerdem darf der Gefahrenbereich jetzt nicht mehr betreten werden.

```
; decide which movement command is executed by checking the movement type
SWITCH COPY_MOVEMENT_TYPE
CASE #KBOT_MVMTTYPE_LIN ; type = 'linear'
LIN COPY_TARGET_POSITION
$OUT[20] = TRUE ; start to decorate cookie with EV3

CASE #KBOT_MVMTTYPE_CIRC ; type = 'circular'
CIRC COPY_SUPPORTING_POSITION, COPY_TARGET_POSITION
$OUT[20] = TRUE ; start to decorate cookie with EV3

CASE #KBOT_MVMTTYPE_PTP ; type = 'ptp'
PTP COPY_TARGET_POSITION
$OUT[20] = TRUE ; start to decorate cookie with EV3

CASE #KBOT_MVMTTYPE_END ; type 'go to home'
LIN COPY_TARGET_POSITION
$OUT[21] = TRUE ; turn on the 'work-done' light
ENDSWITCH
```

Listing 7.24: Wahl der Bewegungsart nach gesetzten Variablen

Je nach Wert der COPY_MOVEMENT_TYPE Variable wird festgestellt welche Bewegung der Roboter fahren soll.

```
; turn off the 'work-in-progress' light
$OUT[22] = FALSE

; purposely revoke permission for roboter to start next movement
; this is done so that the upper-level controller is able to decide when the roboter should move
; (it could be the case that the upper-level controller receives the signal to late)
; ACTIVE AGAIN AFTER TEST
;KBOT_MVMT_STARTSIGNAL=FALSE

; signal upper-level controller that this movement finished
KBOT_READY_FOR_MVMT=TRUE

ENDWHILE
```

```
15 ;FOLD PTP HOME  Vel= 100 % DEFAULT;%{PE}%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:HOME, 3:, 5:100, 7:
       DEFAULT
   $BWDSTART = FALSE
17 PDAT_ACT=PDEFAULT
   FDAT_ACT=FHOME
19 BAS (#PTP_PARAMS,100)
   $H_POS=XHOME
21 PTP  XHOME
   ;ENDFOLD
23
   END
```

Listing 7.25: Nachbereitung setzen der Betriebslampen und Statusvariablen

Nach Beendigung des Verziervorgangs wird die Betriebslampe wieder entsprechend gesetzt und der Roboter fährst zurück auf die vorgegbene HOME-Position.
Zunächst werden die "Velocity (Fließgeschwindigkeit des Materials)" und die Bewegungsdaten kopiert, wie hier im Unterabschnitt 7.2.1 erläutert dient das dazu, dass die Originale einstweilen neu beschrieben werden können. Außerdem werden die Warnleuchten der Industrieanlage geschalten um für alle sichtbar zu machen, dass der "Arbeitsraum (der Bereich in dem der Roboter fahren kann oder darf)" des Roboters im Moment nicht betreten werden darf, da sich die Persons sonst verletzen könnte. (Nähere Sicherheitsvorschriften finden sich in Kapitel 8) Danach folgt eine Fallunterscheidung um heraus zu finden welche Art von Bewegung ausgeführt werden soll. Nun muss die Betriebslampe wieder richtig gestellt und die Variable "KBOT_READY_FOR_MVMT" auf true gesetzt werden. Beides geschieht um zu signaliseren, dass der Roboter nun seine aktuelle Bewegung beendet hat.

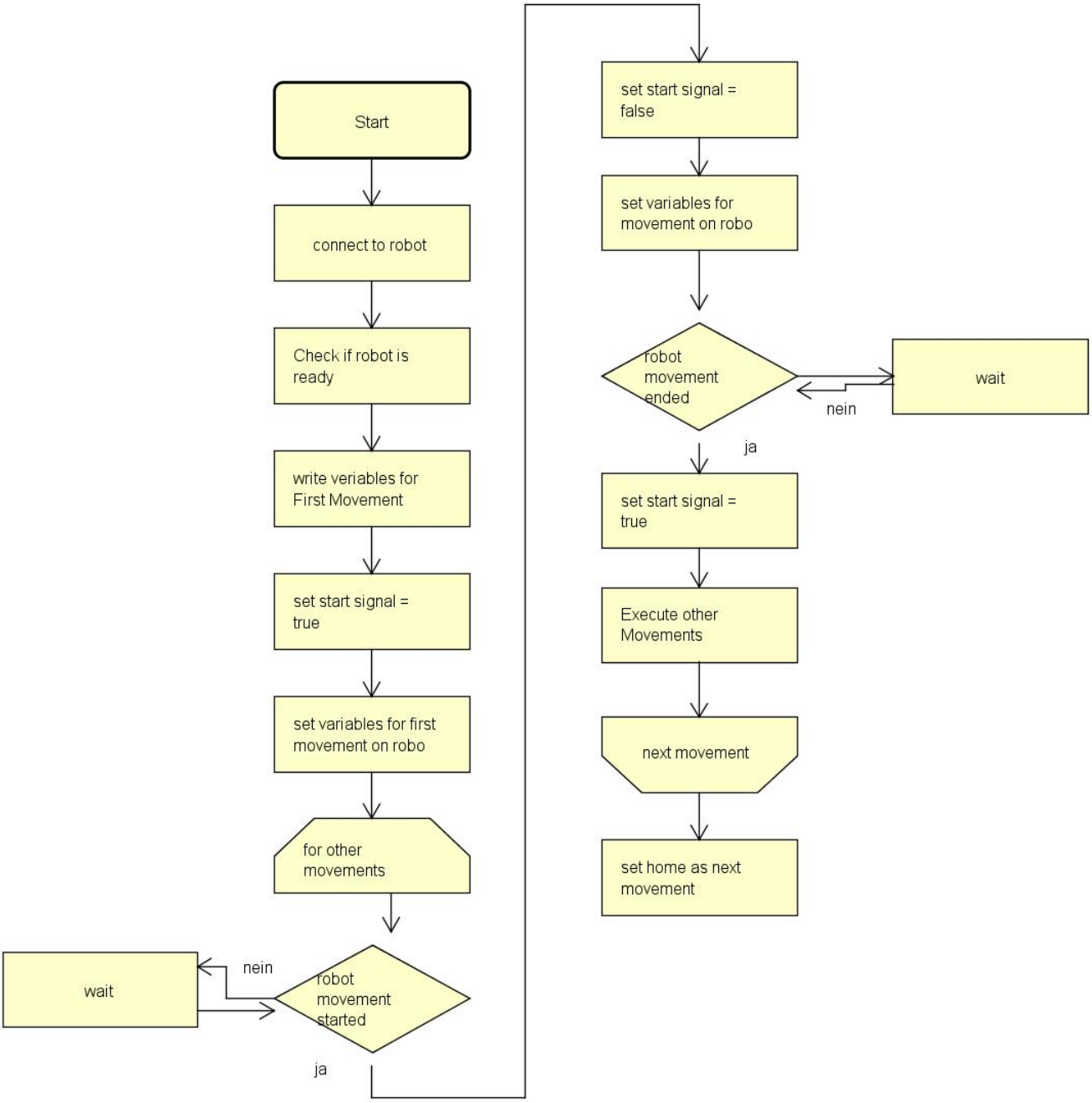


Abbildung 7.6: Ablaufdiagramm Roboterverbindung

Kapitel 8

Robotersicherheit beim Kekse verzieren (CM)

Welche Sicherheitsmaßnahmen sind notwendig wenn ein Bäcker oder eine Bäckerei das keks-o-bot System bei sich verwenden möchte?
Generell ist der Bäcker verpflichtet alle Risiken so weit wie möglich zu minimieren oder eliminieren, gegen alle übrigen Risiken müssen entsprechende Schutzmaßnahmen getroffen und die Mitarbeiter entsprechend geschult werden. Diese und alle weiteren Maschinenrichtlinien finden sich in der Richtlinie 2006/42/EG. [38] Zunächst muss der Arbeitsraum des Roboters von einem Käfig oder einer ähnlich stabilen Vorrichtung umgeben sein. Dadurch können grobe Fehler vom Roboter, oder Unfälle aufgrund von Unachtsamkeiten durch Bediener gering gehalten werden. Abschnitt 8.3 Dazu muss ein Arbeitsbereich Abschnitt 8.2 definiert werden, der die Arbeitsfläche des Roboters und seinen maximalen Schwenkbereich umfasst. Der umzäunte Gefahrenbereich muss zusätzlich noch die maximalen Bremswege umfassen. Als Empfehlung kann sich hier an eine Würfelform gehalten werden, mit den Grundmaßen der Arbeitsfläche und einer Höhe, sodass der Roboterarm maximal auf seine "Homeposition" fahren kann. Um die Absperrung betreten zu können muss eine Tür angebracht werden, die mit einem Öffnungssensor Unterabschnitt 8.3.2 ausgestattet sein muss, damit unerlaubtes Betreten während einer Roboterbewegung registriert werden kann. Weiters müssen ausreichend viele Not-Halt Knöpfe, Abschnitt 8.1 um den Roboter schnell abschalten zu können, vorhanden sein. Zusätzlich kann, muss aber nicht, ein LichtgitterUnterabschnitt 8.3.3 angebracht werden um die Zuführbarkeit von Keksen zum Roboter zu erleichtern. Eine reine Kollisionserkennung Abschnitt 8.4 bietet nicht die notwendige Sicherheit und ist daher nicht ausreichend.
Außerdem müssen die Mitarbeiter die in weiterer Folge direkt mit dem Roboter arbeiten sollen entsprechend eingeschult werden.

8.1 Not-Halt

Die Anlage muss über außreichend Not-Halt Knöpfe verfügen die zu einem sofortigen Abbruch der Roboterbewegung führen. Einer muss außen am Käfig befestigt sein, um von außerhalb erreicht werden zu können. Der äußere Not-Halt wird dann benötigt wenn der Roboter durch falsche Bewegungen einen Sachschaden (niemand befindet sich bei ihm im Käfig - alle Türen und Schranken sind geschlossen) verursachen kann.

8.2 Arbeitsbereich

Dem Roboter muss ein erlaubter Arbeitsbereich definiert werden den er nicht verlassen darf. Würde er dies aufgrund eines Fehlers machen bricht das Programm ab. Generell darf der Arbeitsbereich frei gewählt

werden, es wird allerdings empfohlen einen kubischen (Würfel) zu wählen. Die Grundmaße orientieren sich an Länge und Breite des verwendeten Backblechs, müssen aufgrund des dynamischen Roboterprogramms und der daraus hervorgehenden unvorhersehbaren Bewegung allerdings größter gewählt werden. Außerdem soll die Höhe maximal der gewählten Homeposition des Roboters entsprechen.

8.3 Bedienerschutz

8.3.1 Käfig

Eine Maschine muss so gebaut und betrieben werden, dass sie während ihrer Arbeit keine Personen gefährdet. Ein stabiler Käfig ist hierfür eine einfache und günstige Umsetzungsvariante. So kann der Roboter während des Verziervorganges keine Mitarbeiter in unmittelbarer Nähe verletzen.

8.3.2 Türsensoren

Ein Türsensor ist im Prinzip nichts anderes als ein Signal, das übertragen wird solange die Türe geschlossen bleibt. Öffnet sich diese wird es unterbrochen und dieser Umstand an die Robotersteuerung gemeldet. Sobald diese das Signal, oder das nicht Vorhandensein des Signals, registriert, wird die Roboterbewegung beendet. Öffnet sich die Türe während der Roboter im Automatikmodus verziert, wird das Programm abgebrochen. Um den unerlaubt eingetretenen Bediener nicht zu verletzen. Der Roboter bremst dabei bahntreu.

8.3.3 Lichtgitter

Im Gegensatz zu herkömmlichen Lichtschranken, die aus einem Strahl bestehen, setzt sich ein Lichtgitter[27] aus mehreren, parallel verlaufenden Strahlen zusammen. Es wird durch einen Sensorstreifen und einen LED-Streifen erzeugt. Solange dieses “Gitter”, auch Netz genannt, nicht unterbrochen wird, gilt der Arbeitsbereich als abgesichert.

In der Industrie werden sie vor allem bei automatisierten Prozessen verwendet. Einerseits werden sie in der Fließbandüberwachung verwendet, um die Menge an produzierten Gütern zu protokollieren oder ihre Abmessungen zu überprüfen.

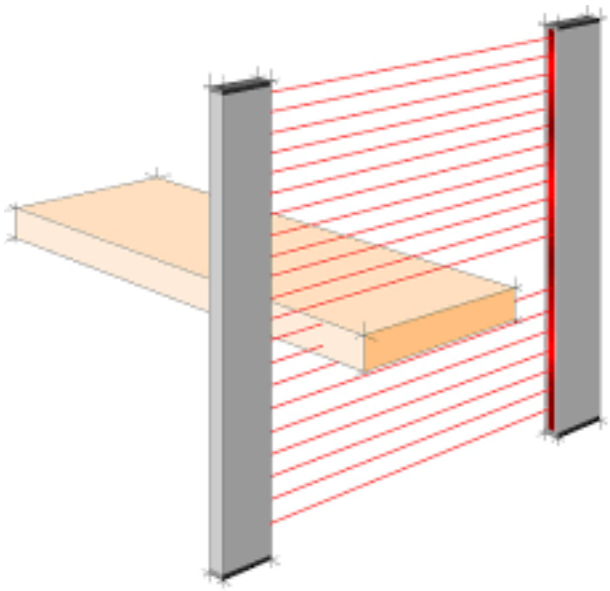


Abbildung 8.1: Lichtgitter zur Werkstück Vermessung/Zählng
Bildquelle: `https://www.lucom.eu/lI-sender-empfaengerleiste-lichtgitter-5mm.html` (abg. am 27.03.2017)

Der interessanteste Anwendungsfall für dieses Projekt ist allerdings die Gefahrenraumüberwachung. Der Bereich, der vom Roboter angefahren werden kann und daher nicht von Personen betreten werden soll, wird an den Zugängen, oder falls keine Absperrungen vorhanden sind, komplett, von Lichtgittern umzäunt. Betritt nun jemand diesen Bereich und unterbricht dadurch das Signal hält der Roboter an.

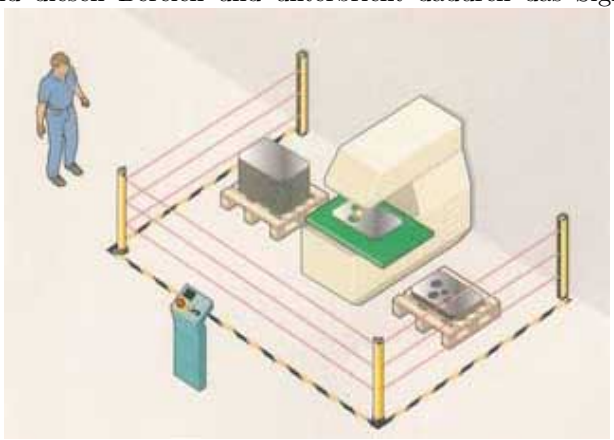


Abbildung 8.2: Lichtgitter zur Bewachung des Arbeitsbereichs
Bildquelle: `http://www.interempresas.net/Deformacion-y-chapa/Articulos/10322-Proteccion-del-acceso-a-maquinas-de-punzonado-estampado.html` (abg. am 27.03.2017)

Generell können Lichtgitter auf 3 Arten für die Absicherung des Verzierungsvorganges verwendet werden:

1. Gefahrenraum Sicherung anstatt eines Käfigs
2. Gefahrenraum Sicherung zusätzlich zu einem Käfig
3. Gefahrenraum Sicherung einer Schleuse für die Kekszufuhr

Der zuvor erwähnte Käfig kann zur Gänze durch Lichtgitter ersetzt werden. Diese Variante wäre um ein Vielfaches teurer und nur dann von Vorteil wenn ständiger Zugang zum Roboter notwendig ist.
Stattdessen ist es möglich das Lichtgitter zu verwenden um "offene Stellen", zum Beispiel Käfig der nur halb hoch ist um das Blech schnell und einfach austauschen zu können, zu sichern. Die Steuerung erkennt

ob die Schranke unterbrochen wurde und ob das gerade vorgesehen ist. Ist dem nicht so, weil entweder der Bediener oder der Roboter einen Fehler machen, bricht die Bewegung ab.
Innerhalb des Käfigs, der durch eine Tür mit Sensoren gesichert ist, kann auch Raum für Vorbereitungen, wie Ablage für Rohkekse und verzierte Kekse zum Trocknen, sein. Die Trennung zwischen Roboter und Bediener kann auch dann über ein Lichtgitter stattfinden, so dass der Bediener zwar manchmal den Käfig betreten, zum Beispiel während der Reinigung oder solange der Roboter auf einen Auftrag wartet, aber nicht das Lichtgitter durchbrechen darf.
Sollte eine Schleuse für Kekse oder Bleche direkt an der Außenseite des Käfigs vorhanden sein können die Lichtgitter auch dazu verwendet werden, sicherzustellen, dass nur etwas mit einer Maximalhöhe von einem Backblech hindurch geschleust wird.
Solange ein Käfig vorhanden ist, muss kein Lichtgitter installiert werden, es kann aber nach eigenem Ermessen des Bäckers und unter Berücksichtigung der Sichehitsbewertung zusätzlich verwendet werden.

8.4 Kollisionserkennung

Moderne Roboter verfügen immer öfter über eine sogenannte “Kollisionserkennung”. Das bedeuted, dass sie eine sensorische Rückmeldung erhalten, sobald sie ein Objekt oder eine Person berühren und daraufhin stoppen. Dafür gibt es mehrere Ansätze:

- Erkennen von Ablenkungen [9]

  Hierbei wird anhand eines Vergleichs von der errechneten Fahrbahn zur tatsächlichen versucht zu erkennen ob der Roboter mit etwas in Berührung gekommen ist.

- Kapazitive Sensorik [20]

  An den beweglichen Roboterteilen befinden sich hier Membranen die bei Berührung ihren Wiederstand ändern und so von der Robotersteuerung wahrgenommen werden können.

- Taktile Sensorik [42]

  Mit dieser Methode wird versucht den menschlichen Tastsinn möglichst getreu nachzuahmen. Es wird ein hochauflösendes Bild der Oberfläche erzeugt.

Generell wird sehr viel auf diesem Gebiet geforscht. Solange hier aber von keiner höheren Sicherheit ausgegangen werden kann, bietet dieses System nicht die notwendige Sicherheit um die automatische keks-o-bot Verzierung zu sichern, da Industrieroboter mit sehr viel Kraft und Geschwindigkeit betrieben werden.

Kapitel 9

Prototyp eines Glasurspenders (MH)

9.1 Verzierungswerkzeug für den Roboter

Obwohl das tatsächliche Glasieren eines Kekses nicht Teil des Projekts ist und bloß exemplarisch mittels Stift und Papier zu beweisen ist, haben wir uns das Ziel gesetzt unsere Arbeit im angemessenem Rahmen vorstellen zu können. Um den Interessenten ein besseres Gefühl für den Nutzen übermitteln zu können ist es allerdings von Vorteil wirklich etwas zu verzieren.

9.1.1 Variante 1: Spritze

Das Auftragen mithilfe einer Spritze zu realisieren liegt nahe, weil es dem Vorgang bei der Ausübung des Patisserie-Handwerks mit Tülle und Beutel am ähnlichsten ist ohne das menschliche Feingefühl zu benötigen.
Vorteile:

- Präziser Auftrag
- Präzise Werkzeugvermessung möglich
- Günstig
- Automatisiertes Nachfüllen möglich

Nachteile:

- Sehr beschränkte Menge (100ml / 50 ml)
- Lineare Bewegung wird benötigt um die Spritze herunter zu drücken

Die beschränkte Füllmenge wäre vernachlässigbar wenn die Vorteile dieser Methode betrachtet werden. Die benötigte linear Bewegung stellte sich allerdings als Hinderniss heraus. Es wurde in Betracht gezogen sich eine fertige Konstruktion auszuleihen die genau dieses Problem mithilfe eines Motors und einer Schraube löst. Abgesehen von den Montageproblemen die sich daraus ergeben hätten und der Tatsache, dass Aufgrund der immer gleichbleibenden Geschwindigkeit die notwendige Evaluierung nicht durchgeführt werden kann, gibt es in dem Spritzenverbau keinen Servomotor (kontrollierbarer Elektromotor) und er kann daher nicht vom KUKA-Roboter, der für den Prototypen genutzt wird, angesteuert werden.

9.1.2 Variante 2: Tubenquetscher

Die nächste Option ist das Befüllen einer handelsüblichen Tube und das Entleeren über einen Tubenquetscher. Auch hierbei handelt es sich um eine sehr kostengünstige Option, die allerdings insofern pro-

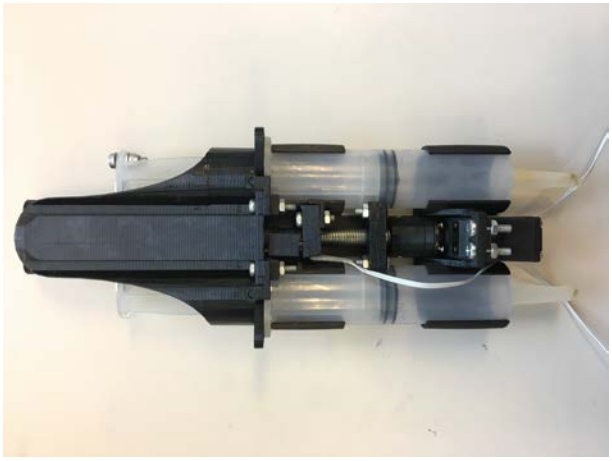


Abbildung 9.1: Spritzenverbau

blematisch ist, als dass sich die z-Werte, also die Höhe der Vorrichtung aufgrund des hinaufdrehens durch den Quetscher beim ausdrücken, während des Ausquetschens ändern und die Tube durch die Bewegung zu sehr schlänkern würde.

Abbildung 9.2: Abbildung eines Tubenquetschers
Bildquelle:
`http://christinamachtwas.blogspot.co.at/2013/04/christinas-ordnungsfimmel.html` (abg. am 27.03.2017)

9.1.3 Variante 3: Pneumatik-Zylinder

Mit Pneumatik-Zylindern kann eine Linearbewegung mittels Luftdruck erzeugt werden. Um sie nutzen zu können, muss der Kolben an dem beweglichen Part des Zylinders montiert werden. Dieser besteht nämlich aus einem "starren" Teil und einem beweglichen Innenteil. Dadurch wird er sich aufgrund des Drucks mitbewegen. Das Gerät wird zwar elektronisch gesteuert, die Druckluft lässt sich, mit dem zur Verfügung stehenden Equipment für den Prototyp, trotzdem nicht präzise genug dosieren um sich für die Aufgabe zu qualifizieren. Außerdem wird dafür ein Kompressor benötigt.

Abbildung 9.3: Abbildung eines Pneumatikzylinders Bildquelle: `http://www.timmer-pneumatik.de/artikel/Pneumatikzylinder/pneumatikzylinder-2.html` (abg. am 27.03.2017)

Variante	Einfach	Automatisiert nachfüllbar	Günstig	Präzise
1 Spritze		X	X	X
2 Tubenquetscher			X	
3 Pneumatik Röhre		X		X
4 Lego Quetscher	X		X	X

Tabelle 9.1: Übersicht Glasurspendervarianten

9.1.4 Variante 4: Lego-Quetscher

[11] Während einer Recherche zu state-of-the-Art Produkten wurde ein Artikel entdeckt, der beschreibt wie Kekse mithilfe eines Lego EV3-Sets verziert werden können. Hier ist die Füllmenge auf die eines handelsüblichen Spritzbeutels begrenzt, sodass er regelmäßig per Hand nachbefüllt beziehungsweise ausgetauscht werden muss.

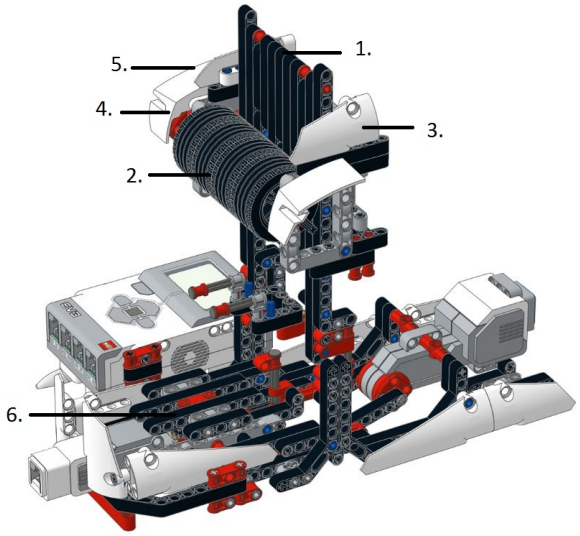


Abbildung 9.4: Abbildung des Verzierungsparts von der Website Bildquelle: `https://www.vodafone.de/featured/digital-life/kloetzchen-fuer-kekse-wie-du-mit-lego-deine-plaetzchen-dekorierst/` (abg. am 27.03.2017)

9.1.5 Gewählte Variante

Aufgrund der Einfachheit und der Tatsache, dass die benötigten Einzelteile zur Verfügung stehen, fiel die Entscheidung auf den "Lego-Quetscher" obwohl er, wie in der Tabelle zu sehen ist, mit der Spritzenvari-

ante gleich auf ist. Hier überzeugt wirklich die einfache Umsetzungsmöglichkeit.

9.1.6 Konzept

Das Gerät kann nach der gegebenen Anleitung nachgebaut werden. Weil diese Konstellation den Anspruchen, in Sachen Stabilität, Präzision und Montage, nicht zur Gänze genügt müssen noch folgende Änderungen am Original vorgenommen werden.

- Veränderte Motorposition um das Gewicht gleichmäßiger zu verteilen
- Untersetzung des Motors mit einem Schneckengetriebe um eine niedrigere Geschwindigkeit zu erreichen. Zusätzlich verhindert die Schnecke dass sich bei Motorstillstand die Quetschrolle durch ihr Eigengewicht dreht.

Abbildung 9.5: Abbildung eine Schneckengetriebes Bildquelle: `https://www.mfg.com/de/manufacturing-knowledge-center/mechanical/schneckengetriebe` (abg. am 27.03.2017)

9.2 Automatisierte Reinigung des Glasurspenders

Durch die Verwendung von verschiedensten Verziermaterialien wird der Glasurspender verschmutzt. Der enthaltene Zucker beziehungsweise die Sahne oder ähnliche Inhaltsstoffe verkleben eine Spritztülle mit der Zeit, falls die Reste nicht, zumindest großteils, entfernt werden. Außerdem können sich auch verschiedene Materialien an der Spitze vermischen was zu einem unreinen Ergebnis führt. Deswegen ist es notwendig sich über eine automatisierte Reinigungsmöglichkeit Gedanken zu machen.

9.2.1 Variante 1: Wasserbad

Hierfür wird der die Spritztülle in ein Wasserbad getunkt um ihn sie zu reinigen. Die Durchführbarkeit scheitert an der Tatsache, dass die Tülle nicht weit genug aus der Halterung herausstehen kann um sie problemlos eintauchen zu können. Es muss der Großteil des Werkzeuges untergetaucht werden, was unschönes Abtropfen zur Folge hat.

9.2.2 Variante 2: Schwammform I

Die Reinigung könnte auch statt finden indem mit dem Ende, also der Tplle über einen flachen Schamm gefahren wird. Dabei wird sie an ihm abgestreift. Außen herum zu kreisen ist nicht möglich da zwischen Spitze und Halterung nicht genügend Platz zur Verfügung steht.

Variante	Einfach	Reinigt innen	Reinigt außen	Reinigt nicht mehr als nötig
1 Wasserbad	X	X	X	
2 Schwammform I	X		X	X
3 Schwammform II	X	X	X	X

Tabelle 9.2: Übersicht Reinigung des Glasurspenders

9.2.3 Variante 3: Schwammform II

Diese Idee baut auf der vorherigen auf und verändert diese dahingegend, dass nun nicht über einen flachen Schwamm sondern über eine Schwammspitze gefahren wird. Sie wurde auf 2 Arten untersucht.

Streichen

Bei der ersten Methode werden 2 Punkte definiert : einer über der Schwammspitze und der zweite so tief, dass die Spitze in die Tülle hineinfährt. Dabei wird die Außenseite nicht gereinigt was zu unschönen Rückständen führen kann.

Drehen

Die zweite Methode basiert auf dem selben Prinzip, mit dem Unterschied, dass ein Punkt definiert wird bei dem die Spitze in der Tülle ist und ein zweiter, der nur eine Drehbewegung um die Z-Achse definiert, so werden auch Überhänge abgerissen.

9.2.4 Gewählte Variante

Insgesamt hat sich die Version mit der Schwammspitze und der Drehbewegung als sinnvollste heraus gestellt, da es nur mit ihr möglich ist die Tülle von Außen und Innen, ohne sie komplett nass zu machen, zu reinigen.

9.2.5 Konzept

Zur Umsetzung muss einerseits eine, möglichst kostengünstige, Halterung für den Schwamm gebaut, andererseits die Bewegung in den Verziervorgang eingebetet werden.
Für die Halterung ist eine einfache Holzplatte (Siehe Abbildung Abbildung 9.6) als Grundlage möglich. An ihr wird der Schwamm mittels einer Klemme (Siehe Abbildung Abbildung 9.7) befestigt.
Die Integration in den Verziervorgang wird nach jeder Shape stattfinden, da sich eine Shape unter anderem über ein bestimmtes Material definiert und des daher sinngemäß ist, vor einem Wechsel das Werkzeug zu säubern.

9.3 Bau des Glasurspenders für den Roboter

Wie in der Planung (Siehe Planungskapitel Absatz Abbildung 9.1.3) erwähnt wird der Glasurspender aus dem “Klötzchen für Kekse”[11] Projekt übernommen. Der Spender besteht aus verschiedenen Teilen.

1. Rückwand
2. Quetschrolle

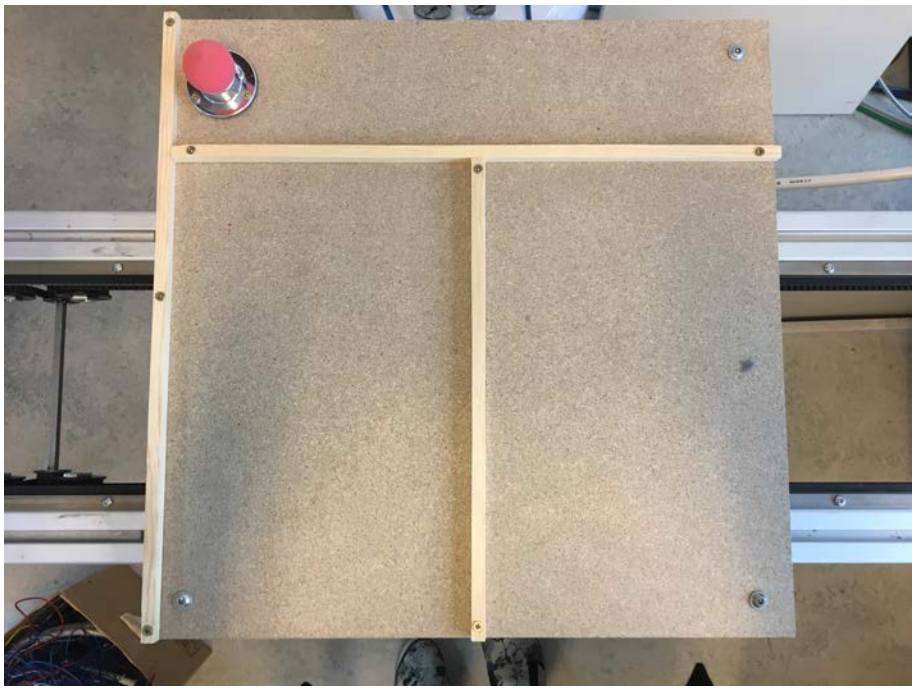


Abbildung 9.6: Fertige Halterung aus Holz mit montiertem Schwamm

3. Halterung rechts
4. Halterung links
5. Motor
6. Boden

Da der Spender am Roboter montiert werden soll wird der Boden nicht gebraucht. Außerdem ist der vorgeschlagene Aufbau zu instabil da der Motor auf einer Seite hängt und so das Gewicht nicht gleichmäßig verteilt ist. Deswegen wird die Konstruktion so adaptiert, dass sich der Motor auf der Rückseite befindet um das Gewicht besser zuverlagern. Außerdem wird eine untersetzung des Motors mit einem Schneckengetriebe erreicht um eine niedrigere Geschwindigkeit zu erzeugen. Zusätzlich verhindert die Schnecke dass sich bei Motorstillstand die Quetschrolle durch ihr Eigengewicht dreht.

Automatisierte Reinigung des Glasurspenders

Um die automatische Reinigung nun durchzuführen müssen zunächste 2 Variablen definiert werden. Wobei der erste Punkt die Position der Reinigungsschwamm-Spitze definiert :

```
DECL E6POS KBOT\_CLEAN\_TIP={X -55.30, Y 30.60, Z 21.88, A 157.27, B 86.37, C -28.39, S 6, T 27}
```

Listing 9.1: Position der Schwammspitze

Und die zweite die Drehung um die Z-Achse hervorruft:

```
1 DECL E6POS KBOT\_CLEAN\_ROTATION={X -45.73, Y 21.91, Z 19.73, A 157.27, B 87.11, C -60.35, S 6, T
      27}
```

Listing 9.2: Drehung um die Spitze

Beide werden nach jeder Shape angefahren, da sich eine Shape über ein Material bzw eine andere Grundeinstellung (z.B. Farbe, Temperatur, etc) definiert. Im KRL Modul wurde die Abfrage um welche Be-

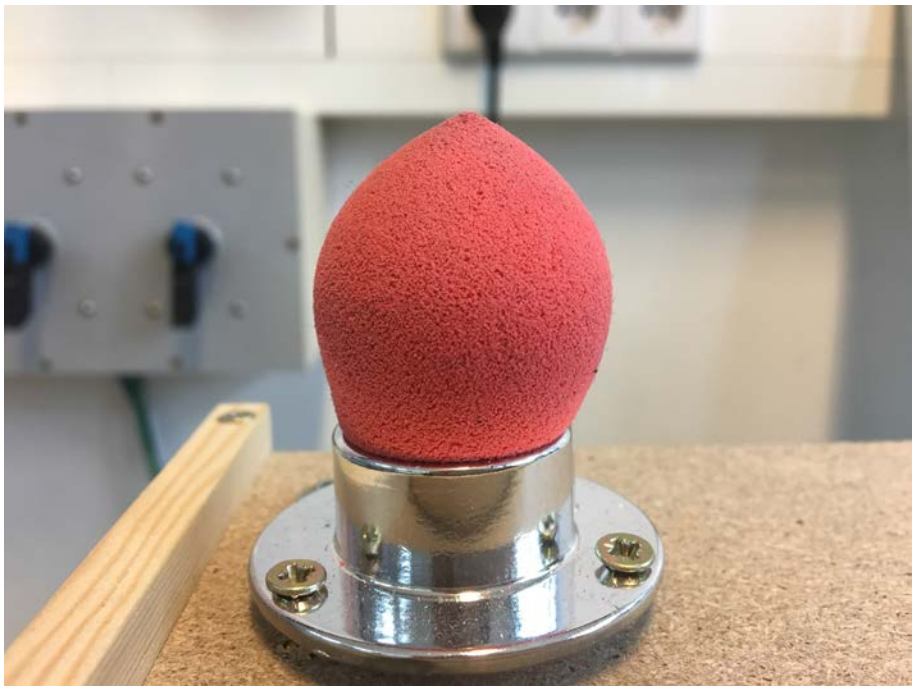


Abbildung 9.7: Befestigung Schwamm

wegung es sich handelt um die Bewegung "KBOT_MVMTTYPE_CL" ergänzt. Wird diese übermittelt fährt der Roboter die oben erwähnten Punkte an.

```
CASE #KBOT\_MVMTTYPE\_CL ;type = 'clean' movement to clean the spout
PTP KBOT\_CLEAN\_TIP
LIN KBOT\_CLEAN\_ROTATION
```

Listing 9.3: Abfrage ob es sich um die Reinigung handelt

In der Java Applikation würden folgende Zeilen nach der Schleife für die Bearbeitung des Movements, aber innerhalb der Shape-Iteration hinzugefügt.

```
List<Movement> shape\_movements = s.getShapeMovements();
Movement shape\_lastmovement = shape\_movements.get(shape\_movements.size()-1);

double xBeginClean = shape\_lastmovement.getXend();
double yBeginClean = shape\_lastmovement.getYend();

Movement clean = new Movement("clean",xBeginClean,yBeginClean);
writeVariablesForMovement(clean);
```

Listing 9.4: Reinigungsvariablen setzen

Die Methode "WriteVariablesForMovement" musste, ebenso wie das KRL-Modul, um die Abfrage ob es sich um eine clean-Bewegung handelt, erweitert werden.

```
/* set KRL variable for movement type */
if(m.getType().equals("linear"))
  robot\_movementType.setValue("KBOT\_MVMTTYPE\_LIN");
else if(m.getType().equals("circularArc"))
  robot\_movementType.setValue("KBOT\_MVMTTYPE\_CIRC");
else if(m.getType().equals("point"))
  robot\_movementType.setValue("#KBOT\_MVMTTYPE\_PTP");
```

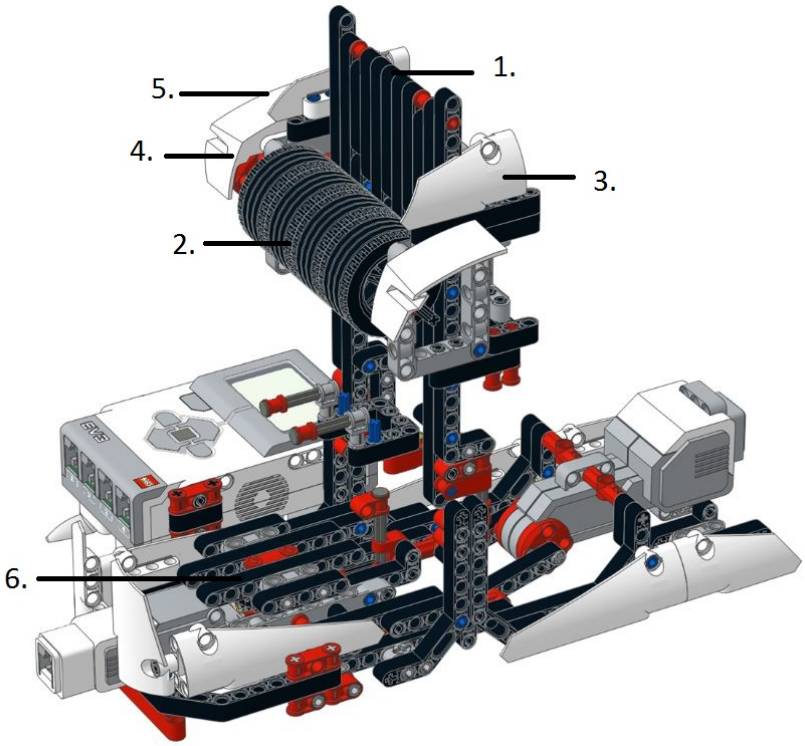


Abbildung 9.8: Abbildung des Verzierungsparts von der Website Bildquelle: `https://www.vodafone.de/featured/digital-life/kloetzchen-fuer-kekse-wie-du-mit-lego-deine-plaetzchen-dekorierst/` (abg. am 27.03.2017)

```
8     else if(m.getType().equals("endmovement"))
        robot\_movementType.setValue("#KBOT\_MVMTTYPE\_END");
10    else if(m.getType().equals("clean"))
       robot\_movementType.setValue("#KBOT\_MVMTTYPE\_CL");
12
  this.robotConnection.writeVariable(robot\_movementType);
```

Listing 9.5: Erweiterte Abfrage

9.4 Ansteuern des Glasurspenders

9.4.1 Senden von Daten vom KUKA-Roboter zum Glasurspender Prototypen

Der KUKA-Roboter kann Output Variablen setzen. Genauso werden auch die Betriebslampen (Siehe Abschnitt 7.2) angesteuert. Nur wird für die Betriebslampen der WAGO Feldbus verwendet. Für die Steuerung des EV3s ist diese Methode aber zu unhandlich weil ein Kabel vom Schaltschrank bis an den TCP geführt werden müsste. Allerdings verfügt das verwendete Robotermodell auch über einen Signlausgänge direkt auf dem Roboterarm.

Im "WorkVisual"[48] kann jetzt ein der entsprechende Ausgang definiert werden. Der Ausgang ist wie in Listing 9.6 beschrieben belegt.

```
1  $OUT[20] sendet eine "1" an den EV3
```

Listing 9.6: Ausgang für Glasurspender Prototyp

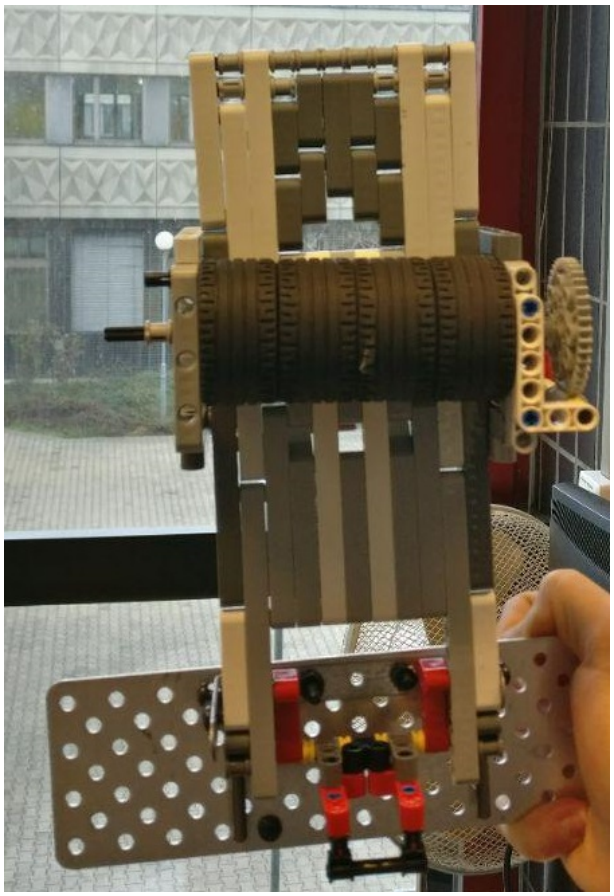


Abbildung 9.9: Glasurspender von vorne

In KRL-Programmen können er dann wie in Listing 9.7 beschrieben angesprochen werden.

```
1 $OUT[20] = TRUE
```

Listing 9.7: Ansprechen des Glasurspender Prototyps

Er wird immer dann, wenn eine "Verzierungsbewegung"beginnt auf TRUE gesetzt.

9.4.2 Einlesen von Signalen am EV3

Der EV3 verfügt über 4 Eingänge an denen verschiedene Sensoren angebracht werden können. Dafür gibt es von Lego Mindstorms verschiedene analog und digital Sensoren[18] zu kaufen. Dazu zählen unter anderem:

- Lichtsensor
- Temperatursensor
- Ultraschallsensor
- Tastsensor

Da dem EV3 nur mitgeteilt werden soll "beginne mit dem Ausquetschen" beziehungsweise "stoppe das Ausquetschen und fahre wieder nach oben" würde sich ein Tastsensor gut eignen. Einerseits ist er digital und überträgt daher nur zwei Zustände und andererseits ist der Aufbau am leichtesten nachzuahmen. Sensoren können im Allgemeinen direkt in der Mindstorms eigenen "Entwicklungsumgebung" per Drag and Drop ausgewählt und abgefragt werden. Hier muss nurmehr der jeweilige Eingang, auf dem der Sensor

Abbildung 9.10: Glasurspender von der Seite

angeschlossen ist, ausgewählt werden. Es gibt auch eine vordefinierte “warte auf Sensor” Funktion, die den Ablauf unterbricht bis etwas eingelesen wird.(Siehe Abbildung Abbildung 9.11)

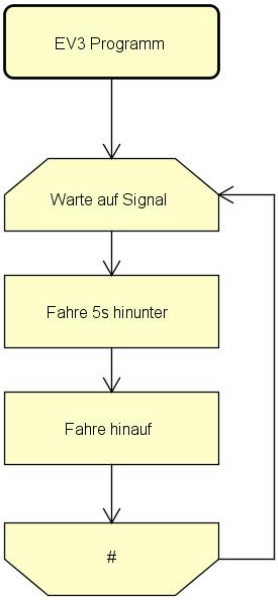


Abbildung 9.11: Ablauf des EV3 Programms

9.4.3 Bau eines Sensors

Es wird also ein Tastsensor nachgebaut, wobei das "Schließen" durch den KUKA-Roboter simuliert wird. Das passiert nicht indem er tatsächlich auf einen Knopf drückt sondern indem der neue Sensor so mit dem Stecker verkabelt wird, dass er dann, wenn das KRL-Programm diesen Ausgang auf HIGH schaltet, schließt.

Anforderungen [7]

Prinzipiell erkennt der EV3 nur die für ihn vorgesehenen Sensoren. Überprüft wird das über einen Spannungteiler, der vom EV3 registriert wird und der so den Sensortypen feststellt. Um also einen eigenen Schalter bauen zu können muss darin auch dieser identifizierende Spannungsabfall nachgebildet werden.

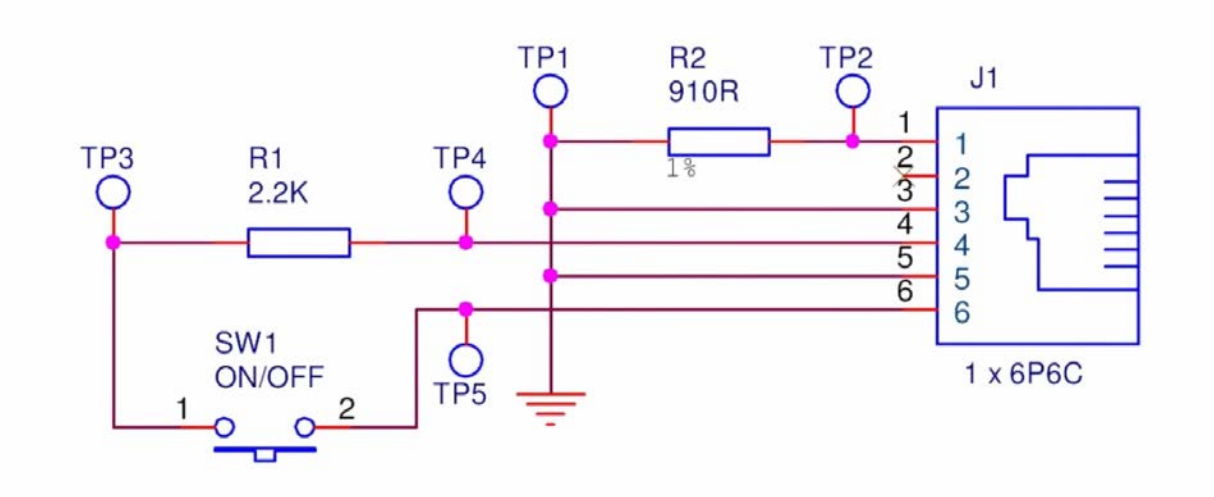


Abbildung 9.12: Abbildung des Verzierungsparts von der Website Bildquelle: `https://www.youtube.com/watch?v=3rIK1XeKHEU` (abg. am 27.03.2017)

Umsetzung mit Steuerung durch den KUKA-Roboter

Dieser Aufbau (Siehe Abbildung 9.12) wird also nachempfunden und danach mit dem KUKA-Roboter verbunden. Der EV3 kann zur Veranschaulichung provisorisch befestigt werden.

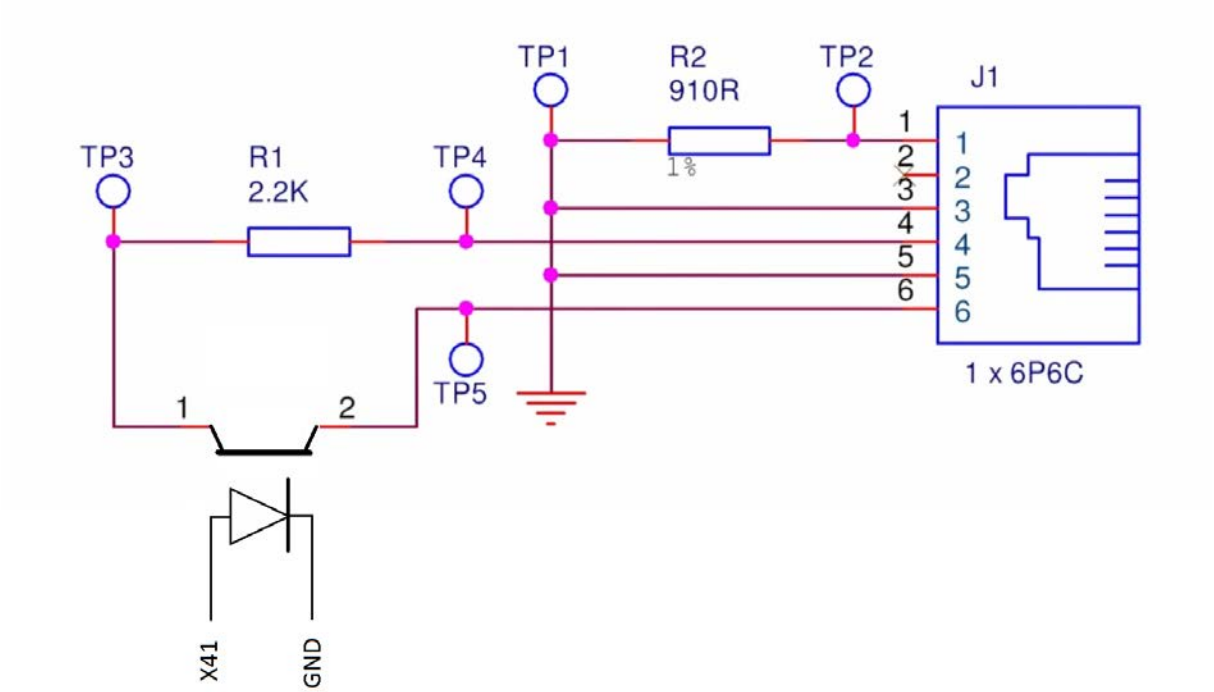


Abbildung 9.13: Abbildung des Verzierungsparts von der Website Bildquelle: `https://www.youtube.com/watch?v=3rIK1XeKHEU` (abg. am 27.03.2017)

Zu diesem Zweck wird ein "Optokoppler"[34] anstatt eines Schalters an den EV3 angeschlossen.(Siehe Abbildung Abbildung 9.13) Die Verbindung findet über einen fotosensitiven Transistor statt. Auf der anderen Seite des Optokopplers befindet sich eine LED die, wenn der KUKA-Roboter den Eingang auf HIGH schaltet, auf den Transistor leuchtet und so den Stromfluss auf der anderen Seite ermöglicht. Verbunden wird er am KUKA-Roboter über den X41 Eingang direkt am Roboterarm. [12]

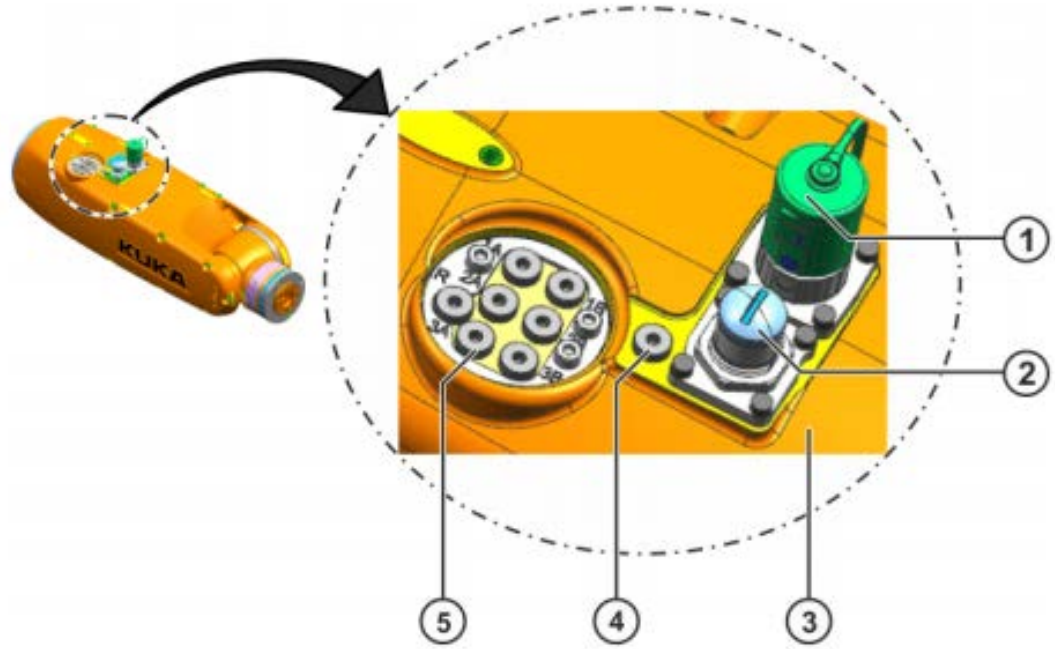


Abb. 6-8: Schnittstelle A4, Beispiel

1	Anschluss X41	4	Luftleitung AIR2
2	Anschluss XPN41	5	Luftanschlüsse
3	Zentralhand		

Abbildung 9.14: Abbildung des Verzierungsparts von der Website Bildquelle: `https://github.com/mhauer-tgm/Syt/blob/master/Spez_KR_AGILUS_sixx_de.pdf` (abg. am 27.03.2017)

Der KUKA-Roboter sendet über diesen Ausgang ein Signal mit 24V, der EV3 hingegen arbeitet bloß mit 5V. Der verwendte Optokoppler muss deswegen mit einer großen Eingangsspannung arbeiten können. Normalerweise werden Optokoppler dazu verwendet Schaltung mit einer großen Spannung durch kleine Spannungen zu schließen, nicht anders herum wie hier. Dieser kann mit bis zu 24V befeuert werden und damit bis zu 48V schalten. Daher müssen hier keine weiteren Verstärker oder Regler verbaut werden. Nun wird der Schalter durch einen Input, der vom Roboter ausgeht, ersetzt.

Kapitel 10

Installation & Betrieb

10.1 Installation der Komponenten

10.1.1 Systemvoraussetzungen (RG)

Die Keksobot-Komponenten haben unterschiedliche Anforderungen an die auf dem System installierte Software. Als Betriebssystem wird dabei für alle Komponenten eine Distribution von Linux empfohlen, obwohl die Komponenten plattformunabhängig sind, weil unter diesem Betriebssystem das meiste Testing stattfand.

Komponente	**Voraussetzungen**
Grafikverarbeitung	• Java SE 8
Auftragsumwandlung	• Java SE 8 • Python 2.7 • Python-Modul ttfquery (Version 1.0.5)
Produktionsschnittstelle	• Java SE 8 • JOpenShowVar[0]
Web-Auftritt	• PHP 7.1 • MySQL 5.6 • GD 2.2.3 • Composer 1.3.2

Tabelle 10.1: Systemvoraussetzungen für Keksobot-Komponenten

[0] vom Keksobot-eigenen Fork der Software, siehe `https://github.com/rgschi-tgm/JOpenShowVar`, Commit-Hash `3ca76a4`

Hinweis: Neben den Keksobot-Komponenten gibt es beim verwendeten Referenz-Roboter auch Voraussetzungen. Siehe dafür Abschnitt 10.2.

10.1.2 Vorgangsweise (RG)

Das volle Keksobot-Softwarepaket kann entweder durch das angebotene Installations-Script installiert werden, oder auf manuellem Weg.
Die einzelnen Keksobot-Komponenten können auf getrennten Rechnern betrieben werden. Eine mögliche Trennung der Komponenten ist beispielsweise:

- Rechner 1: Web-Auftritt (Benutzerschnittstelle + HTTP-Backend)
- Rechner 2: Grafikumwandlung
- Rechner 3: Auftragsumwandlung
- Rechner 4: Produktionsschnittstelle

Eine selektive Installation ist mit dem Installations-Script aber nicht möglich, weshalb dann eine manuelle Installation nach den folgenden Anweisungen bevorzugt werden sollte. Ansonsten ist es auch möglich das Installations-Script auf jedem Rechner auszuführen, dann aber werden redundante Daten gelagert.

Das Installationsscript ist als `install-kbot.sh` im Wurzelverzeichnis des entpackten Keksobot-Dateiarchivs zu finden. Von diesem Pfad ausgehend werden die ausführbaren Dateien der Komponenten und deren Ressource-Dateien in die für Software vorgesehenen Orte des Linux-Dateisystems kopiert. Der Anhaltspunkt für die Installationsorte ist der Filesystem Hierarchy Standard (FHS) [3, Kap. 4].

- `/usr/bin`
  enthält ausführbare Programmdateien. Beispielsweise ist die Datei `kbot-decorationPreprocessor.jar`.
- `/usr/lib/keksobot`
  enthält Ressource-Dateien, wie etwa die XML-Schemadefinitionen für die Input-Verifizierung zwischen den Komponenten. Außerdem beinhaltet das Verzeichnis die Konfigurationsdateien des Init-Systems *systemd*, mit denen Keksobot-Komponenten gestartet und verwaltet werden.
- `/etc/keksobot.d`
  enthält Konfigurationsdateien, darunter beispielsweise `Keksobot.properties` und `kukarobot.properties`.
- `/var/www/keksobot`
  Alle für die Kundenschnittstelle notwendigen Dateien diverser Web-Technologien werden unter diesem Verzeichnis verwaltet.
- `/var/log`
  Die Keksobot-Komponenten zeichnen Ereignisse standardmäßig an dieser Stelle auf. Mögliche Dateien sind zum Beispiel `kbot-preprocessor.0.log` oder `kbot-converter.0.log`. Der Pfad für Logdateien ist allerdings konfigurierbar. Dafür siehe Abschnitt 10.4.

Das Installationsscript kopiert Keksobot-Systemdateien an die gerade vorgestellten Orte im Dateisystem. Es bindet es die sogenannten Servicedateien für Keksobot-Komponenten in das Init-System *systemd* ein. Es handelt sich hierbei um die Konfigurationsdateien, mit denen Keksobot-Komponenten gestartet und deren Betrieb verwaltet wird. Eine Funktionalität, die durch *systemd* hergestellt wird, ist beispielsweise der automatische Neustart der Keksobot-Komponente im Falle eines Programmabsturzes. Das Installationsscript kopiert die Servicedateien für *systemd* an den vorgesehenen Ort und aktiviert sie, sodass ab dem nächsten Neustart bei jedem Systemstart die Keksobot-Komponenten automatisch aktiviert werden. Nach der Installation bleiben die Komponenten allerdings bis zum nächsten Start gestoppt, um anschließende manuelle Handgriffe zu ermöglichen.

Hinweis: Die Installation mittels Script endet nach dessen erfolgreicher Ausführung mit einem Neustart.

Das Installations-Script übernimmt das Verschieben der Dateien für die ausgewählten Dateien und bereitet bis zu einem bestimmten Grad die erste Inbetriebnahme vor. Abhängig von der Umgebung, in der die Installation stattfindet, müssen aber manuelle Konfigurationsschritte vorgenommen werden. Falls die vorgestellten Szenarios zutreffen, sind die genannten Maßnahmen zu ergreifen.

Hinweis: die vorgestellten Schritte sind auch bei der gänzlich manuellen Installation notwendig, falls sie zutreffen.

Wann?	**Maßnahme**
Keksobot-Komponenten werden voneinander entfernt betrieben	Auf allen Maschinen in `Keksobot.properties` die Kommunikationsparameter definieren (siehe Abschnitt 10.4).
bei manueller Installation, oder bei nicht-unixbasierten temen	Environment-Datei des Systems um die Variable `KEKSOBOT` erweitern (bei Linux z.B. `/etc/environment`), die den Dateisystem-Pfad des Konfigurations-Verzeichnisses angibt.
wenn kein KUKA-Roboter verwendet wird	In diesem Fall muss die Roboterkommunikations-Komponente selbstständig entwickelt werden. Sie muss Aufträge über die in `Keksobot.properties` festgelegten Kommunikationsparameter entgegennehmen.

Tabelle 10.2: notwendige manuelle Installations-Handgriffe in bestimmten Szenarios

Wenn die Ausführung des Installationsscripts fehlschlägt, oder eine manuelle Durchführung bevorzugt wird, ist auch das möglich. Der Inhalt der folgenden Verzeichnisse muss an die definierten Zielorte kopiert werden:

- `exported/*` —> `/usr/bin`
- `configuration/*` —> `/etc/keksobot.d`
- `resource/*` —> `/usr/lib/keksobot`
- `website/*` —> `/var/www/keksobot`

Die *systemd*-Servicedateien werden wie in Listing 10.1 aktiviert und sind ab dem nächsten Neustart gültig.

```
1 systemctl enable /usr/lib/keksobot/systemd-services/*
```

Listing 10.1: manuelle Aktivierung der Systemd-Servicedateien

10.2 Vorbereitung der Produktionsebene (MH)

10.2.1 TCP Port 7000 freigeben und KUKAVARPROXY.exe starten

Das KUKAVARPROXY.exe Programm ist eine Grundanforderung um die Kommunikation zwischen Produktionsschnittstelle und Roboter zu ermöglch. Damit es funktionieren kann muss der Port 7000 freigegeben sein. Um den Port freizugeben muss in den `KUKA-Einstellungen -> Inbetriebnahme -> Netzwerkkonfiguration -> Erweitert -> Reiter "NAT" -> Port 7000 fuer TCP` hinzugefügt werden.

Um das KUKAVARPROXY-Programm zustarten wird über `KUKA-Einstellungen -> Anzeige -> Service -> HMI minimieren` die Windows-Umgebung freigeschalten (Mindest-Benutzerlevel ist Experte). Der USB-Stick mit der sich darauf befindende `KUKAVARPROXY.exe` wird an die Robotersteuerung angesteckt. Im Windows Dateiexplorer das Programm an eine geeignete Stelle verschieben. Das Programm ausführen. Es sollte ein Überwachungsmonitor des Socketprogramms in Form einer GUI gestartet werden, der auch die aktiven Verbindungen anzeigt. Falls die GUI nicht starten sollte, muss die Ausführung der EXE über die CMD erneut versucht werden.
Das Programm horcht jetzt aktiv auf den freigegebenen Port 7000 und empfängt alle Daten die an ihn gesendet werden.

10.2.2 Globale Variablen zum Datenaustausch

Dafür wird die globale Konfigurationsdatei unter `KRC:\R1\Steu\$config.dat` editiert. Um die Datei bearbeiten zu können, muss mindestens das Rechtelevel auf Experte gesetzt sein. Am Ende der Konfigurationsdatei ist Platz für eigene Variablen-Definitionen. Der folgende Inhalt wird eingefuegt:

```
1 DECL BOOL KBOT_READY_FOR_MVMT = TRUE
  DECL BOOL KBOT_MVMT_STARTSIGNAL = FALSE
3 DECL INT KBOT_MVMT_TYPE = -1
  DECL E6POS KBOT_START_POSITION = (x 0, y 0, z 0)
5 DECL E6POS KBOT_TARGET_POSITION = (x 0, y 0, z 0)
  DECL REAL KBOT_VELOCITY = 0.0
7 DECL E6POS KBOT_SUPPORTING_POSITION = (x 0, y 0, z 0)
```

10.2.3 Konfiguration der Betriebslampen um den Status der Verzierung anzeigen zu können

Die statusanziegende Betriebslampe ist mit einem digiteln Output-Modul (Wago 750), das über den Kuka Extensionbus via EtherCat von der Robotersteuerung geschalten werden kann, verbunden.

Mit der Projektierungssoftware WorkVisual von KUKA wewrden die Aus- und Eingänge des Roboters konfiguriert. Im Folgenden soll nur ein kurzer Überblick über die Vorgehensweise gegeben werden. Eine detaillierte Beschreibung findet sich in dem Konfigurations-Protokoll[37] aus dem Vorjahr. Grundlegende Informationen zur Konfguration von Ein- und Ausgänge finden sich in der Workvisual-Anleitung [49].

- Betriebslampe mit Digital Output-Modul verkabelt
- WorkVisual geöffnet, Roboterkonfiguration einlesen ueber Line Inteface X65
- Extension Bus hinzugefügt, Feldbus Device Description File eingelesen
- Feldbus konfiguriert (Ausgänge vom Modul zu Eingänge vom Roboter)
- neue Konfiguration aus WorkVisual auf den Roboter geladen

Für die Lampen sind die Ausgänge 21 (grün), 22 (gelb) und 23 (rot) vorgesehen.

10.3 Vorbereitung der Webkomponente (CM)

10.3.1 Betriebsystem

Wie im Abschnitt Unterabschnitt 10.1.1, erwähnt sind alle Komponenten von Keks-o-bot planformunabhänig. Im Rahmen der Diplomarbeit wurde auf *Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-45-generic x86_64)* gesetzt. Die Ubuntu-Community ist sehr präsent, was bei möglichen Fehlern sehr hilfreich sein kann. Der Nachteil von Ubuntu ist, dass nicht auf Stabilität sondern Aktualität gesetzt wird. Das bedeutet, dass die zur Verfügung gestellten Pakete immer aktuell sind, was allerdings compatiblitäts Probleme versuchen kann. Im Echtbetrieb wäre einen andere Linux Distribution wie zum Beispiel Debian zu empfehlen.

10.3.2 Installation von PHP7 und Composer

Da Laravel ein PHP-Framework ist muss PHP[35], eine weitvebreitete Webprogrammiersprache, in einer unterstützten Version (ab PHP 5.6) installiert werden. Laravel bietet viele verschiedene Möglichkeiten an um das Framework zu installieren. Die einfachste und deswegen auch gewählte Methode ist via Composer[5], welcher zuerst installiert werden muss.

```
1 sudo apt-get install curl php7.0-cli php7.0-mcrypt git
  curl -sS https://getcomposer.org/installer | sudo php -- --install-dir=/usr/local/bin --filename=
      composer
```

Listing 10.2: Installieren von Composer und PHP7

Nach der Installation können Composer und PHP-Interpreter global auf dem System verwendet werden. Dies wird dadruch bewirkt, dass die Composer- und PHP-binary zum Systempfad hinzugefügt werden. Um sicherzustellen, dass die Installation erfolgreich war kann ein *composer -v* beziehungsweise ein *php -v* ausgeführt werden. Im Falle einer erfolgreichen Installation wird die Version in der Konsole ausgegeben. Bei Composer werden zusätzlich die verfügbaren Befehle angezeigt.

10.3.3 Installation von MySQL

MySQL[31] ist ein relationales Datenbankmanagementsystem welches es in zwei Versionen gibt (opensource und Enterprise). Die opensource Version darf ausschließlich für nicht kommerzielle Tätigkeiten verwendet werden. Bei der kommerziellen enterprise Version gibt es keine Einschränkungen. Im Rahmen der Diplomarbeit wurde auf den Kauf der enterprise Version verzichtet, da in absehbarer Zeit kein Markteintritt geplant ist.

Um MySQL zu installieren[45] müssen folgende Schritt durchgeführt werden:

```
  #Updaten des Package Indexes
2 sudo apt-get update

4 #Installation des MySQL-Servers
  sudo apt-get install mysql-server
6
  #Konfiguration und Installation (Konsolen-Dialog) von MySQL
8 sudo mysql_secure_installation
  sudo mysql_install_db
10
  #Der MySQL Server soll jedes mal beim starten des Server gestartet werden
```

```
12 systemctl enable --now mysql.service
```

Listing 10.3: Installation von MySQL

10.3.4 Installation und Konfiguration von Laravel

Im vorherigen Abschnitt wurden alle Grundbausteine für eine erfolgreiche Laravel Installation gelegt. Mit folgendem Befehl wird Laravel, im aktuellen Ordner, installiert:

```
composer create-project --prefer-dist laravel/laravel .
```

Listing 10.4: Installieren von Laravel via Composer

Um das Framework verwenden zu können muss ein systemspezifischer *Environment-Key* generiert werden. Dieser wird im Environment-Datei gespeichert. Die Environment-Datei ist eine Ressource, auf welche von der gesamten Applikation aus zugegriffen werden kann. Es wird deswegen unter anderem für globale Variablen, Securitytokens und Verbindungsdaten für die Datenbank (kurz DB) verwendet. Der Environment-Key wird in weiterer Folge für jeden Verschlüsselungsvorgang und zum Compilieren der PHP-Dateien verwendet.

```
1 cp .env.example .env
  php artisan key:generate
```

Listing 10.5: Installieren von Laravel via Composer

Die endgültige Installation kann nun mittels *php artisan serv*, ein von Laravel zur Verfügung gestellter Webserver, gehostet und getestet werden.

Für den Live betrieb wird es empfohlen diesen Befehl in einem *screen* zu starten. Dieser sorgt dafür, dass der Webserver, auch nach schließen der Konsole, im Hintergrund weiterläuft.

```
  #Installieren von Screen
2 sudo apt-get install Screen

4 #Hosten der Website in einem Screen
  screen -A -m -d -S <name des screens> <Befehl z.B.: php artisan serv>
6
  #Anzeigen aller aktiven Screens
8 screen -ls

10 #Beenden des Screens
   screen -X -S <Name des screens> kill
```

Listing 10.6: Installieren von Screen und starten des build-in Webservers von laravel

Durch das Eintragen der Zugangsdaten für die Datenbank in das Environment-File wird der Zugriff auf die Datenbank ermöglicht.

```
1 DB_CONNECTION=mysql
  DB_HOST=my-server-ip-address
3 DB_PORT=3306 //default port mysql
  DB_DATABASE=mydatabase
5 DB_USERNAME=myusername
  DB_PASSWORD=mysecret
```

Listing 10.7: Eintragen der Zugangsdaten für die Datenbank in dem Environment-File

10.4 Konfiguration

10.4.1 Keksobot-Komponenten (RG)

In der Datei `Keksobot.properties` werden Konfigurationsparameter gesetzt, die Keksobot-eigene Komponenten beeinflussen. Keksobot-Komponenten lesen die Konfigurationen aus dieser Datei aus und passen ihr Programm zur Laufzeit daran an. Die Tabelle zeigt eine beispielhafte Konfiguration mit allen derzeit verwendeten Konfigurations-Schlüsseln und ihre Bedeutung:

KBOT_WORKBENCH_MAXIMUM_X=800	Breite der Arbeitsfläche in mm, wird für das Vorverarbeiten von Verzierungsgrafiken benötigt
KBOT_WORKBENCH_MAXIMUM_Y=600	Höhe der Arbeitsfläche in mm
KBOT_DECORATION_PREPROCESSOR_HOST=127.0.0.1	IP-Adresse der Grafikverarbeitung
KBOT_DECORATION_PREPROCESSOR_PORT=60210	TCP-Port der Grafikverarbeitung
KBOT_ORDER_CONVERTER_HOST=127.0.0.1	IP-Adresse der Auftragsumwandlung
KBOT_ORDER_CONVERTER_PORT=60220	TCP-Port der Auftragsumwandlung
KBOT_ORDER_EXECUTION_HOST=127.0.0.1	IP-Adresse der Produktionssschnittstelle
KBOT_ORDER_EXECUTION_PORT=60230	TCP-Port der Produktionssschnittstelle
KBOT_LOG_TARGET=/var/log	Zielort für Logdateien der Keksobot-Komponenten
KBOT_ORDER_CONVERTER_DEFAULTFONT=Arial	Standard-Schriftart, die für Text in Verzierungsgrafiken verwendet wird, dessen Schriftart nicht installiert ist

Tabelle 10.3: Eine beispielhafte Keksobot.properties Datei

10.4.2 Verbindung zum Roboter (MH)

Für die Referenzimplementierung der Keksobot-Roboterkomponente, die mit einem KUKA-Roboter arbeitet, existiert eine Konfigurationsdatei namens `kukarobot.properties` (ebenfalls zu finden unter `/etc/keksobot.d`). Die Tabelle 10.4 zeigt eine Beispielkonfiguration.

ROBOT_CONNECTION_HOSTNAME=192.168.42.130	Die IP-Adresse, die zur Steuerung des Roboters führt
ROBOT_CONNECTION_PORT=7000	Der TCP-Port, der zur Steuerung des Roboters führt
ROBOT_MOVEMENT_DEFAULT_Z=50	Der Abstand zur Arbeitsfläche, mit dem der Roboter den Glasurspender über den Keks führt (in mm).
ROBOT_MOVEMENT_DEFAULT_A=177.23	siehe Hinweis `ROBOT_MOVEMENT_DEFAULT_xxxxx`
ROBOT_MOVEMENT_DEFAULT_B=82.22	siehe Hinweis `ROBOT_MOVEMENT_DEFAULT_xxxxx`
ROBOT_MOVEMENT_DEFAULT_C=-2.86	siehe Hinweis `ROBOT_MOVEMENT_DEFAULT_xxxxx`
ROBOT_MOVEMENT_DEFAULT_S=6	siehe Hinweis `ROBOT_MOVEMENT_DEFAULT_xxxxx`
ROBOT_MOVEMENT_DEFAULT_T=27	siehe Hinweis `ROBOT_MOVEMENT_DEFAULT_xxxxx`
ROBOT_HOMEPOSITION_X=0	Die X-Koordinate des Home-Punktes, der nach Beenden der Keksverzierung angefahren wird.
ROBOT_HOMEPOSITION_Y=0	Die Y-Koordinate dieses Endpunkts.
ROBOT_HOMEPOSITION_Z=400	Der Abstand des Roboters zur Arbeitsfläche, wenn er am Home-Punkt verweilt (in mm).

Tabelle 10.4: Eine beispielhafte kukarobot.properties Datei

Hinweis zu `ROBOT_MOVEMENT_DEFAULT_xxxxx`: Die Werte `A,B,C,S,T` wurden der aktuellen Ausrichtung des Roboters entnommen, als der exemplarische Glasurspender in einer senkrecht zur Arbeitsfläche stehende Lage befand (= seine Arbeitsposition). Diese Parameter sind notwendig, um einen Bewegungsbefehl aus dem *Keksobot Robot Neutral Format* zu vervollständigen und an den Roboter übertragen zu können.

10.5 Steuerung der Komponenten (RG)

Im Umfang des Keksobot-Installationsarchivs werden folgende Servicedateien angeboten und können in den in Punkt 10.8 und Listing 10.9 vorgestellten `systemctl`-Befehlen angegeben werden.

- für die Grafikverarbeitung: `kbot-decorationPreprocessor.service`
- für die Auftragsumwandlung: `kbot-orderConverter.service`
- für die Produktionsschnittstelle: `kbot-orderExecutor.service`
- für den Web-Auftritt: `kbot-website.service`

```
cd /usr/lib/keksobot/resource/systemd-services && systemctl enable *
cd /usr/lib/keksobot/resource/systemd-services && systemctl start *
cd /usr/lib/keksobot/resource/systemd-services && systemctl stop *
cd /usr/lib/keksobot/resource/systemd-services && systemctl status *
```

Listing 10.8: Systemd-Befehle, um bei allen Keksobot-Komponenten gleichzeitig:

1. den Start bei jedem Bootvorgang zu konfigurieren
2. den sofortigen Programmstart einzuleiten
3. den sofortigen Programmhalt einzuleiten
4. den momentanen Status abzufragen

Ist es hingegen gewünscht, eine spezifische Keksobot-Komponente zu starten (`start`), zu stoppen (`stop`) bzw. ihren Status zu prüfen (`status`), kann von einem beliebigen Verzeichnis aus der folgende Befehl ausgeführt werden:

```
systemctl <Befehl> <Name der Servicedatei einer Keksobot-Komponente>
```

Listing 10.9: Beliebige Keksobot-Komponente verwalten

Die besondere Eigenschaft des *systemd* Init-System ist, dass es einen Programmabsturz wahrnimmt und darauf je nach Konfiguration reagieren kann. In den Servicedateien für Keksobot ist definiert, dass nach einem Absturz die Komponente neu gestartet werden soll. Für eine beispielhafte Servicedatei siehe Listing 10.10.

```
[Unit]
Description=Keksobot decoration preprocessor daemon
After=network.target

[Service]
EnvironmentFile=/etc/environment
WorkingDirectory=/usr/lib/keksobot
ExecStart=/usr/bin/java -jar /usr/bin/kbot_decorationPreprocessor.jar
Restart=on-failure
RestartSec=1
SuccessExitStatus=143

[Install]
WantedBy=network.target
```

Listing 10.10: Eine Keksobot-Servicedatei (Beispiel: Grafikverarbeitung)

Hinweis: Die Klausel `SuccessExitStatus=143` ist notwendig, weil Java bei einem Programmstop von außen (also dem Init-System bei einem `systemctl stop`-Befehl) mit einem Exitcode der Nummer 143 reagiert. Dieser Exitcode wird ansonsten fälschlicherweise als Absturz interpretiert.

10.6 Aufnahme neuer Kekse & Verzierungsgrafiken (RG)

Als Administrator des Keksverzierungsdienstes können neue Basiskekse sowie neue Verzierungsgrafiken in den Katalog geladen werden, die bei Kunden bei neuen Aufträgen zur Auswahl stehen.

Der Vorgang beginnt mit der Angabe von allgemeinen Informationen. Bei neuen Basiskeksen wird angegeben:

- die Bezeichnung für den Kekses (für Kunden als *Keksart* angezeigt)
- die Formkategorie des Kekses (*Keksform*)
- der Grundpreis des Kekses
- die enthaltenen Allergene

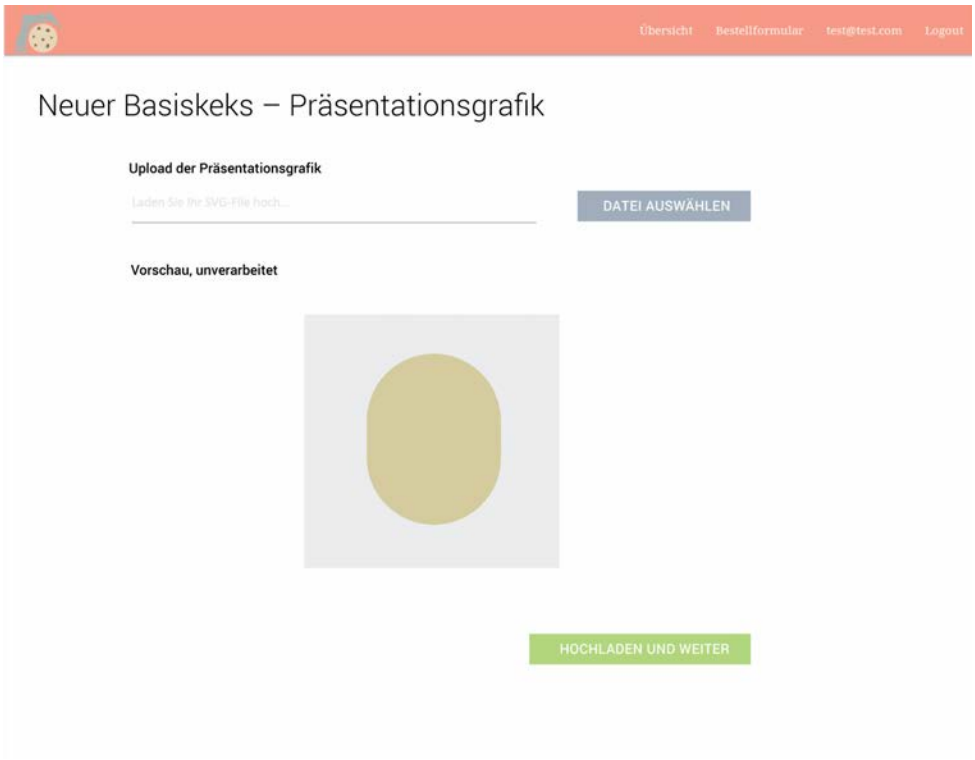
Bei neuen Verzierungsgrafiken hingegen wird nur der Anzeigetext für den Webshop angegeben.

Im nächsten Schritt lädt der Administrator eine Abbildung seines Kekses bzw. seiner Verzierung hoch. Das geschieht im selben Dateiformat, mit dem Kunden im Bestellvorgang ihre eigene Verzierungsgrafik angeben können, also im Dateiformat SVG. Nach dem Upload findet eine Vorverarbeitung durch die Grafikverarbeitung statt, bei der die Grafik an das Keksobot-System angepasst wird. Besteht ein Problem mit der hochgeladenen Datei, wird ein Hinweis ausgegeben.

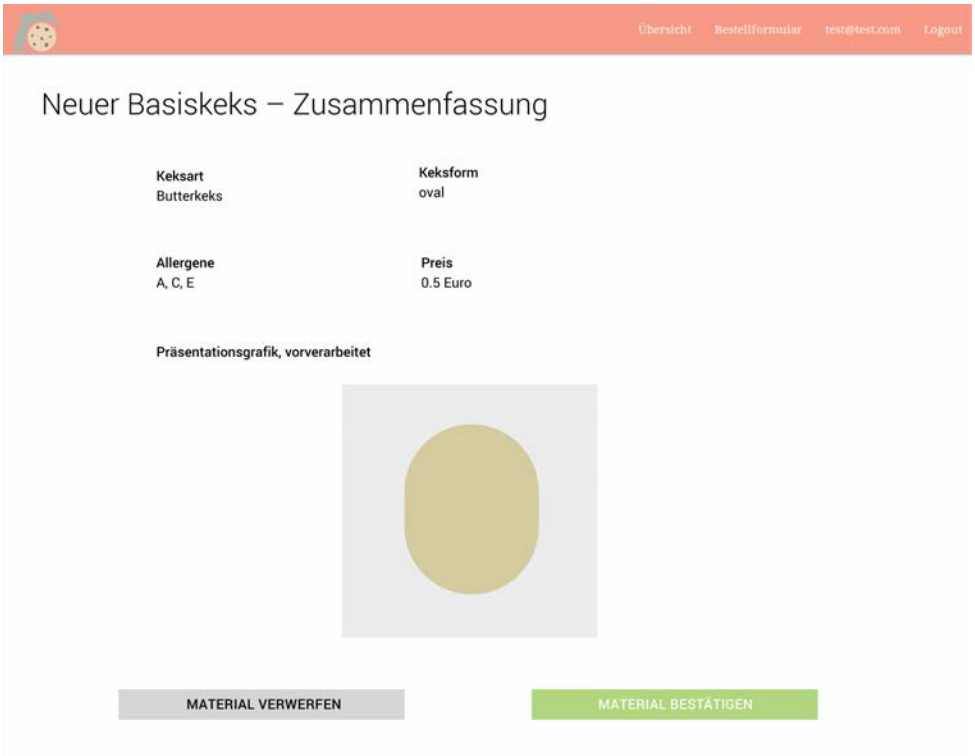
Im letzten Schritt wird die gesammelte Information zusammengefasst dargestellt und die verarbeitete Grafik angezeigt. Das macht eine Verifizierung des Verarbeitungsergebnisses möglich. Ist die angezeigte Information korrekt, wird der Vorgang mit dem Druck auf die Schaltfläche abgeschlossen. Der Keks bzw. die Verzierung ist nun für neue Bestellungen auswählbar.

(a) Schritt 1: Allgemeine Infos

Abbildung 10.1: Benutzerschnittstelle für das Einlesen neuer Kekse



(b) Schritt 2: Upload der Präsentationsgrafik



(c) Schritt 3: Zusammenfassung und Verifikation

10.7 Aufnahme neuer Glasurmaterialien (RG)

Als Administrator des Keksverzierungsdienstes kann ein neues Material in den Katalog aufgenommen werden. Mehrere Schritte müssen dafür durchlaufen werden:

1. Allgemeine Informationen angeben (Bezeichnung, Preis, Allergene, repräsentative Farbe)
2. Materialanforderungen finden (Testverzierungen mit Testmotiven)
3. Gefundene Materialanforderungen angeben
4. Neues Material bestätigen und in Katalog aufnehmen

Hinweis: Der Vorgang wird in Abbildung 10.3 grafisch beschrieben.

Im ersten Schritt werden Informationen angegeben, die nach Aufnahme des Materials im Bestellvorgang für den Kunden sichtbar sind. Eine repräsentative Farbe wird angegeben, um dem Kunden im Bestellvorgang die optische Umsetzung seiner Verzierungsgrafik, mit seinen ausgewählten Glasurmaterialien, als Vorschau anzeigen zu können. Dabei sollte die Farbe so nahe wie möglich dem tatsächlichen Material entsprechen.

Im zweiten Schritt werden die materialspezifischen Anforderungen ermittelt, die das Material bei der Keksverzierung mit Robotern an die Produktionsumgebung stellt. Das sind:

- die Maximalgeschwindigkeit, die der führende Roboter mit diesem Material abfahren darf
- die Pumpkraft, mit der das Material aufgetragen wird
- Heizung aktiv? (optional, standardmäßig deaktiviert)
- Gebläse aktiv? (optional, standardmäßig deaktiviert)

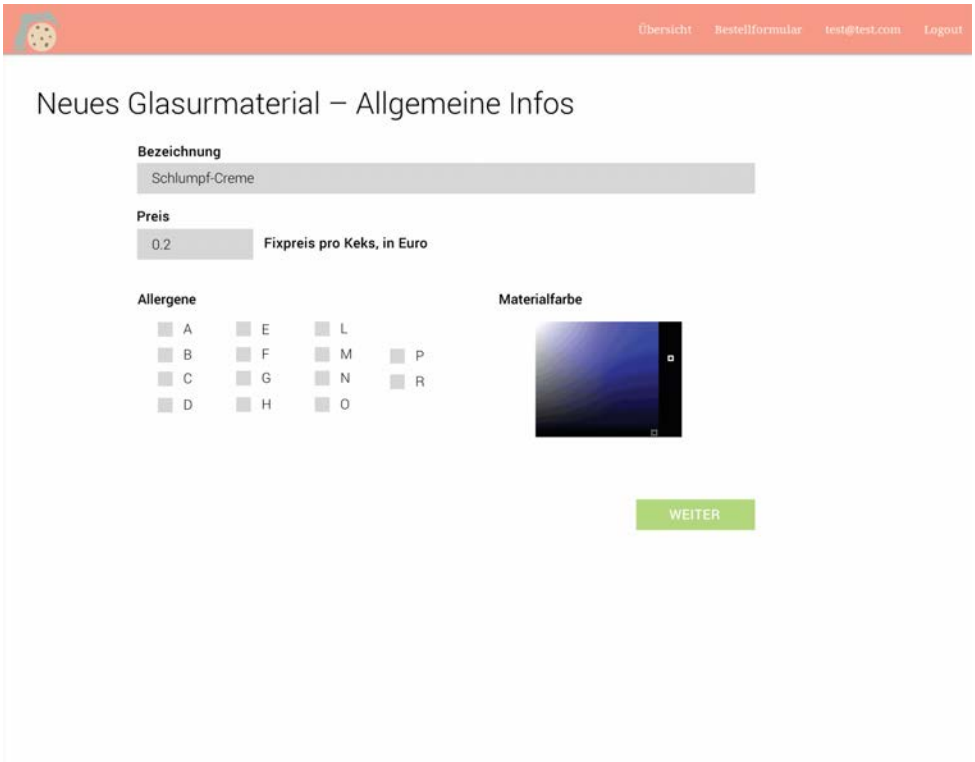
Um diese Materialanforderungen zu finden, wird ein Testmotiv ausgewählt und durch Betätigen der Schaltfläche *Testdurchlauf* ein Testauftrag mit dem ausgewählten Testmotiv in die Produktionsebene eingespielt. Der Testdurchlauf startet, nachdem der aktuelle Keksverzierungs-Auftrag (wenn vorhanden) abgeschlossen wurde, als nächster Auftrag. Der Vorgang wird über die Webseite so lange wiederholt, bis die Materialanforderungen gefunden wurden. Dann wird die Schaltfläche *Test abschließen* betätigt, woraufhin der nächste Schritt beginnt.
Im dritten Schritt werden die gefundenen Materialanforderungen bekanntgegeben.

Anschließend folgt nach der Bestätigung dieses Formularschrittes eine Gesamtübersicht zu den angegebenen Glasurmaterial-Daten. Sind diese korrekt, kann das Material in den Katalog aufgenommen werden. Ab diesem Zeitpunkt ist das Material automatisch für neue Keksverzierungs-Aufträge verfügbar.

10.8 Installation von Schriftarten für die Grafikverarbeitung (RG)

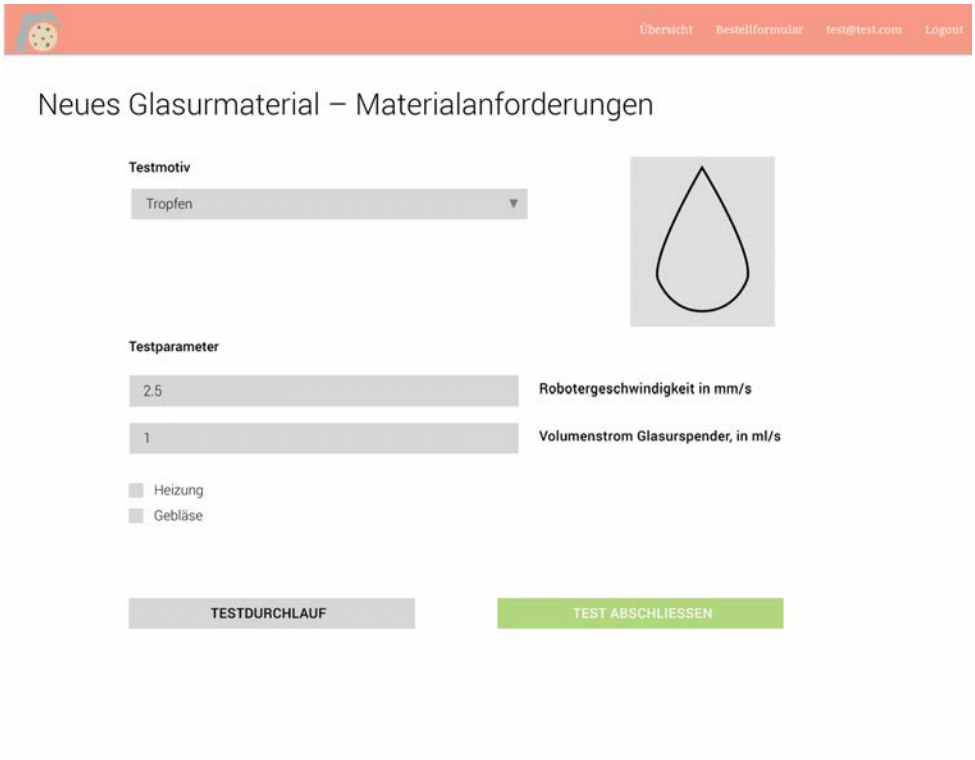
Wie in Unterabschnitt 6.3.3 technisch beschrieben wurde, ist es möglich die Texte aus Verzierungsgrafiken in den gewünschten Schriftarten vom Roboter abfahren zu lassen. Dafür muss die gewünschte Schriftart aber im Keksobot-Schritartenverzeichnis in einem bestimmten Format vorhanden sein. Für die Installation einer neuen Schriftart in den Keksobot-Katalog wird daher ein Hilfsprogramm zur Verfügung gestellt.

Das Hilfsprogramm zur Installation neuer Schriftarten für Keksobot muss auf genau dem System ausge-

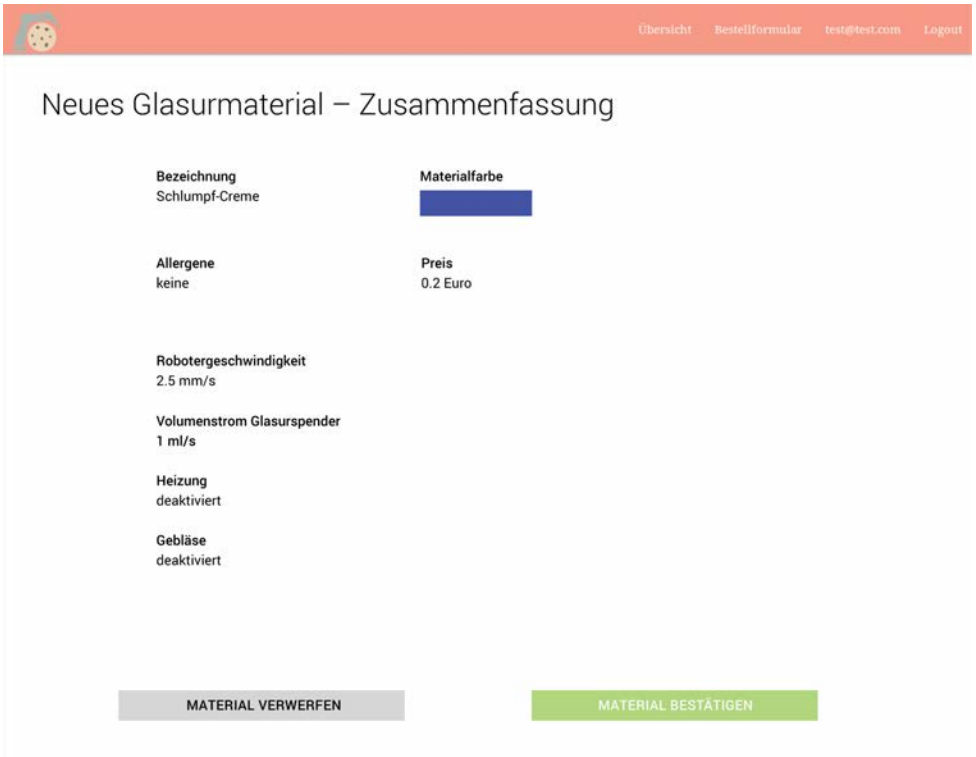


(a) Schritt 1: Allgemeine Infos

Abbildung 10.3: Benutzerschnittstelle für das Einlesen neuer Glasurmaterialien



(b) Schritt 2: Testdurchläufe



(c) Schritt 3: Materialanforderungen

führt werden, auf dem die Umwandlungs-Komponente für Verzierungsgrafiken installiert wurde. Das Programm kann über den Befehl `kbot-fontinstall` aufgerufen werden. Über die Kommandozeilenparameter wird der Pfad zur Schriftart-Datei vom Dateityp `TTF` angegeben (siehe Listing 10.11). Das Hilfsprogramm installiert die angegebene Schriftart dann im benötigten Format in das Keksobot-Schriftartenverzeichnis.

```
kbot-fontinstall <Dateipfad zur Schriftart>
```

Listing 10.11: Aufruf des Hilfsprogrammes für neue Schriftarten

10.9 Umgang mit fehlerhaft verarbeiteten Aufträgen (RG)

Im Laufe des automatischen Verarbeitungsprozesses, der nach dem Absenden eines neuen Kundenauftrags gestartet wird, kann es unter Umständen zu einer Unterbrechung dieses automatischen Prozesses kommen. Manche Fehler bestehen temporär und ergeben sich aus äußeren Einflüssen, Mögliche Fehlerszenarios sind

- Netzwerkfehler: temporär (z.B. Überlastung, kurzzeitiger Hardware-Ausfall, kurzzeitige Fehlkonfiguration) oder permanent (z.B. Hardware-Ausfall ohne Ausfallsicherheit)
- Konfigurationsfehler (z.B. unzureichende oder inkorrekte Angaben)
- Programmfehler (z.B. Fehlverarbeitung, unbekannter Fehler ist aufgetreten)

In einem Fehlerfall wird die korrekte Verarbeitung eines neuen Auftrages verhindert. Das bedeutet, dass es zum Beispiel nie zu einer Umwandlung in das anlagenneutrale Roboteranweisungs-Dokument kam, oder aber der Auftrag nicht in die Warteschlange für die Produktionsanlage eingefügt wurde.

10.9.1 Problemanalyse

Wann immer ein Fehler bei der automatischen Auftragsverarbeitung aufgetreten ist, besteht der Wunsch, die Ursache des Fehlers zu finden. Mit Logbuch-Einträgen versuchen die Keksobot-Komponenten bei der Analyse eines Fehlers zu unterstützen.
Die Logbuch-Einträge beinhalten entweder die Auftrags-Nummer, bei der die Lognachricht entstanden ist, oder wenn zu diesem Zeitpunkt im Bestellvorgang noch keine Auftrags-Nummer vergeben wurde eine eindeutige Identifikationsnummer für die Bestellungs-Instanz. Mit diesen eindeutigen Nummern ist es möglich den Verarbeitungs-Verlauf für einen beliebigen Auftrag nachvollziehen zu können, und somit zu sehen, an welcher Stelle ein Fehler aufgetreten ist.

Mit der Kenntnis darüber, an welcher Stelle der Verarbeitungsprozess unterbrochen wurde, kann nun entschieden werden, um welche Art von Fehler es sich handelt. Wenn der letzte Logbuch-Eintrag einen Kommunikationsfehler anzeigt, ist es notwendig die Erreichbarkeit zwischen den Keksobot-Komponenten zu prüfen oder fehlerhafte Konfigurationen zu berichtigen.
Tritt der Fehler allerdings bei den selben Gegebenheiten (selbe Keksverzierung, selber Kekstyp, ...) immer wieder auf, wurde möglicherweise ein systematischer Fehler entdeckt. Dabei handelt es sich dann um einen Programmfehler in einen oder mehreren Keksobot-Komponenten, die bei Software nicht ausgeschlossen werden kann. Die Informationen des betroffenen Auftrags und die zugehörigen Logbuch-Einträge helfen bei Reproduzierung, der Analyse und letztendlich der Behebung des Problems.

Der Keksobot-Administrator hat in der Benutzerschnittstelle die Möglichkeit, sich Aufträge, die nicht automatisch verarbeitet werden konnten, auflisten zu lassen. Ein Auftrag ist erst vollständig automatisch verarbeitet worden, wenn er zur Warteschlange der Auftrags-Ausführung hinzugefügt wurde. Die Liste gibt mittels der angegebenen Zeitstempel Auskunft darüber, an welcher Stelle ein Fehler im Verarbeitungsprozess auftrat.

Zustand	**mögliche Fehlerursache**	**Fehleranalyse**
Umwandlung-Zeitstempel fehlt	1. Kommunikationsproblem zwischen Komponenten 2. Programmfehler in Umwandlungs-Komponente	• für 1. → Logbuch der Benutzerschnittstelle verwenden • für 2. → Logbuch der Umwandlungs-Komponente verwenden
Warteschlange-Zeitstempel fehlt	1. Kommunikationsproblem zwischen Komponenten 2. Programmfehler in Auftragsausführungs-Komponente	• für 1. →Logbuch der Benutzerschnittstelle verwenden • für 2. →Logbuch der Auftragsausführungs-Komponente verwenden

Tabelle 10.5: Bedeutung der Ausgabe aus "fehlerhafte Aufträge anzeigen"

Hinweis: Fehler während des Bestellvorganges, die vor dem Absenden der Bestellung aufgetreten sind, werden nicht in der Auflistung für fehlerhafte Aufträge angezeigt. Der Grund ist, dass zu diesem Zeitpunkt noch kein Auftrag besteht. Solche Fehler werden aber dennoch in den jeweiligen Logbüchern registriert, sind jedoch ausserhalb der Logbücher nicht sichtbar. Statt der Auftrags-Nummer wird in den Logbuch-Einträgen die sogenannte *Session-ID*, d.h. eine eindeutige Referenznummer eines Bestellvorgangs, verwendet. Damit ist die Rückverfolgbarkeit eines Fehlers möglich. Der Unterschied bei unvollständigen Aufträgen ist aber, dass die Bestelldaten automatisch verworfen werden. Somit ist ein aufgetretener Fehler erklärbar, aber nicht mit den selben Bestelldaten reproduzierbar.

Nachdem die Fehlerquelle gefunden wurde, wird mit der eindeutigen Identifikationsnummer des Auftrags oder der Bestellungs-Instanz nach Logbuch-Einträgen gesucht. Wie diese Logbücher aufzurufen sind, wird in Unterabschnitt 10.9.3 beschrieben.

10.9.2 Erneuten Verarbeitungsversuch einleiten

Als Administrator ist es in der Benutzerschnittstelle möglich, sich fehlerhaft verarbeitete Aufträge auflisten zu lassen. Die Bedeutung der angezeigten Daten wurde bereits in Tabelle 10.5 beschrieben. Einzeln pro Auftrag ist es dem Administrator möglich, einen weiteren Versuch zur Auftragsverarbeitung einzuleiten. Die Benutzerschnittstelle zeigt den Erfolg oder einen erneuten Fehlschlag nach kurzer Zeit an.

Hinweis: Es ist durchaus möglich, dass bei einem erneuten Fehlschlag der Fehler nun an einer anderen Stelle auftritt. Deshalb ist es wichtig den Listeneintrag des erneut fehlgeschlagenen Auftrags nochmals zu prüfen.

10.9.3 Logbücher

Die Logbücher der Keksobot-Komponenten sind als Datei im Logbuch-Dateiverzeichnis vorhanden. Standardmäßig ist das Verzeichnis `/var/log` dafür vorgesehen, der Speicherort kann aber konfiguriert worden sein (siehe Abschnitt 10.4).

Komponente	**Logbuch-Dateiname**
Grafikverarbeitung	`kbot-preprocessor.x.log`
Auftragsumwandlung	`kbot-orderConverter.x.log`
Produktionsschnittstelle	`kbot-orderExecution.x.log`
Web-Auftritt	`kbot-website.x.log`

Tabelle 10.6: Dateinamen der Logbücher der Keksobot-Komponenten

Hinweis: Im Dateinamen steht der Platzhalter `x` für die laufende Nummer der Logbuch-Dateien, wobei die Aktualität mit steigender Zahl abnimmt. `kbot-preprocessor.0.log` ist die aktuellste Logdatei.

Kapitel 11

Conclusio & Outlook

11.1 Lessons learned

Die Aufgabenstellung einer Diplomarbeit kann im ersten und zweiten Moment sehr einschüchternd wirken. Wo angefangen werden soll und auf welche Aufgaben zunächst das Hauptaugenmerk gelegt wird ist niemandem so recht klar. Es hat sich herausgestellt, dass es am wichtigsten ist einfach anzufangen. Selbst wenn es sich im Nachhinein als eine suboptimale Aufteilung herausstellt. Am Besten wird mit einer kleinen Teilaufgabe begonnen und dann Stück für Stück weiter gearbeitet. Denn das einzige was einen nicht weiter bringt ist es, nicht anzufangen, weil es zu schwierig oder umfangreich sein könnte.
Auch wenn nicht aus Furcht sondern aus Faulheit prokrastiniert wird kann das verheerende Folgen haben. Zu Beginn des Jahres war die Motivation in den Teams recht rar gesät, was wohl vor allem daran lag, dass auch der alltägliche Schuldruck noch nicht präsent war. In den meisten Fällen hat das dazu geführt, dass parallel zu der stressigsten Zeit was Schularbeiten und Tests angeht auch irrsinnig viel an der Diplomarbeit zu arbeiten war. Große Aufgaben sollten also eher gleich, wenn noch Zeit und Ressourcen vorhanden sind, angegangen werden.
Sollte sich die Arbeitsmenge doch nicht unterbringen lassen ist es unabdingbar diesen Umstand zu kommunizieren. Das einzig Falsche was gemacht werden kann ist zu versuchen diesen Umstand zu verheimlichen. Im Endeffekt wissen es ohnehin alle, aber umso eher man darüber spricht, umso eher kann eine Lösung gefunden werden.
Es kann nichts Schlimmeres passieren als, dass die Auslastung im Team so hoch ist, dass eine Lastverteilung nicht mehr möglich ist. Na gut, dann geht hald nichts weiter aber wenigstens wissen alle was Sache ist und es kann mit diesem Umstand weiter geplant werden. Im besseren Fall ist der Zusammenhalt im Team so stark, dass eine gemeinsame Lösung gefunden und das Projekt noch zu einem zufriedenstellenden Ausgang geleitet werden kann. Es muss aber auch ein Ende der Arbeit geben. Bei so großen Aufgaben oder Projekten lässt sich immer noch etwas verbessern, noch ein Feature einbauen, noch etwas dokumentieren. Irgendwann muss das Gesamtwerk aber von einer gewissen Distanz betrachtet und gesagt werden: das reicht jetzt. Ansonsten wird die Diplomarbeit von einer Bereicherung zu einer ungeheuren Last die keine Freude mehr bereiten kann. Mit diesem neu gewonnenem Wissen kann die nächste Diplomarbeit ja nur mehr besser werden!

11.2 Zukunftserwartungen

Bei allen fix terminierten Projekten ist es notwendig, aus dem breiten Pool an vorstellbaren Funktionen eine an den gegebenen Zeitrahmen angepasste Schnittmenge auszuwählen. Dabei wurden auch sinnvolle Funktionalitäten notwendigerweise vernachlässigt. In der aktuellen Softwareversion befinden sich Stellen, an denen die metaphorischen Schrauben noch ein wenig angezogen werden könnten.

Der Webauftritt könnte durch Multilingualität und Internationalisierung für eine Erweiterung der Kundenlandschaft sorgen. Außerdem würde die Anpassbarkeit des Website-Designs für Kunden des Softwarepakets eine Möglichkeit zur Individualisierung bieten. Der Katalog für Verzierungsgrafiken könnte um weitere Verzierungen und um eine Filterfunktion erweitert werden.
Das Webshop-System ist im Projekt als Prototyp vorgesehen worden, weshalb vor dem öffentlichen Betrieb noch Vorbereitungen empfehlenswert sind, wie etwa den Bestellvorgang gegen die lokalen gesetzlichen Rahmenbedingungen zu prüfen, und ein Sicherheits-Auditing durchführen. Im Webshop-Prototyp ist der Preis von Produkten noch nicht flexibel gestaltbar. Funktionen wie diverse Rabatte (Mengen- oder zeitabhängiger Rabatt) oder Gutscheincodes würde das Webshop-System für Unternehmer aufwerten. Um das Keksobot-Softwarepaket für den Enterprise-Sektor tauglich zu machen, könnte die Unterstützung von Enterprise Resource Planning (ERP)-Systemen entwickelt werden.

Das zentrale Backend könnte mehrere Dateiformate für kundeneigene Verzierungsgrafiken unterstützen, um die potenzielle Kundenlandschaft zu erweitern. Zum Beispiel wäre der Dateityp EPS ebenfalls geeignet, weshalb sich bei der Abwägung der zweite Platz ergab. Die Grafikverarbeitung könnte sich außerdem die Unterstützung von Rastergrafiken annehmen. Weniger technikaffine Kunden werden dann eher zu der Funktion der kundeneigenen Keksverzierung greifen, die derzeit abschrecken könnte, weil das Dateiformat SVG der breiten Masse weniger geläufig ist.
Der Grafikverarbeiter unterstützt derzeit keine Transformationen, somit auch keine rotierten Grafikobjekte. Die Verwendung der Funktion durch die Kunden ist aber durchaus realistisch.

Bei der Auftragsumwandlung wurden Optimierungen erkannt, mit denen Bézier-Kurven effizienter verarbeitet werden könnten. Derzeit werden sie durch eine fixe Anzahl an kurzen Linearbewegungen angenähert, die in gleichbleibendem Abstand definiert werden. Kurze Kurven benötigen aber eine geringere Anzahl an Linien, um die Kurve gut anzunähern. An Bereichen der Kurve mit weniger Information (dem “Tal”) darf die Annäherung ungenauer sein als am Ort mit viel Information (dem “Hügel”).

Nie zuvor hatten wir so wenig Zeit, um so viel zu tun.

Franklin D. Roosevelt

Kapitel 12

Anhang

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration {
    /**
    * Run the migrations.
    *
    * @return void
    */
    public function up() {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('firstname');
            $table->string('lastname');
            $table->string('email')->unique();
            $table->string('password');
            $table->string('stripe_customer_id')->nullable();
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
    * Reverse the migrations.
    *
    * @return void
    */
    public function down() {
        Schema::drop('users');
    }
}
```

Listing 12.1: Create-User Migration

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreatePasswordResetsTable extends Migration {
    /**
```

```
9       * Run the migrations.
        *
11      * @return void
        */
13      public function up() {
            Schema::create('password_resets', function (Blueprint $table) {
15              $table->string('email')->index();
                $table->string('token')->index();
17              $table->timestamp('created_at')->nullable();
            });
19      }

21      /**
        * Reverse the migrations.
23      *
        * @return void
25      */
        public function down() {
27          Schema::drop('password_resets');
        }
29  }
```

Listing 12.2: Create-Password-Reset Migration

```
    <?php
2
    use Illuminate\Support\Facades\Schema;
4   use Illuminate\Database\Schema\Blueprint;
    use Illuminate\Database\Migrations\Migration;
6
    class CookieShapes extends Migration {
8       /**
        * Run the migrations.
10      *
        * @return void
12      */
        public function up() {
14          Schema::create('cookie_shapes', function (Blueprint $table) {
                $table->increments('id');
16              $table->longtext('shape_svg');
                $table->timestamps();
18          });
        }
20
        /**
22      * Reverse the migrations.
        *
24      * @return void
        */
26      public function down() {
            Schema::drop('cookie_shapes');
28      }
    }
```

Listing 12.3: Cookie-Shape Migration

```
1   <?php

3   use Illuminate\Support\Facades\Schema;
```

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class Svg extends Migration {
    /**
    * Run the migrations.
    *
    * @return void
    */
    public function up() {
        Schema::create('cookie_decorations', function (Blueprint $table) {
            $table->increments('id');
            $table->longtext('decoration_data');
            $table->timestamps();
        });
    }

    /**
    * Reverse the migrations.
    *
    * @return void
    */
    public function down() {
        Schema::drop('cookie_decorations');
    }
}
```

Listing 12.4: Cookie-Decorations Migration

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class material extends Migration {
    /**
    * Run the migrations.
    *
    * @return void
    */
    public function up() {
        Schema::create('materials', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('color'); //hex
            $table->string('color_category'); //human color "name" RED;BLUE
            $table->text('allergens');
            $table->integer('price');
            $table->timestamps();
            $table->unique(['name','color_category']);
        });
    }

    /**
    * Reverse the migrations.
    *
    * @return void
    */
```

```
    public function down() {
        Schema::drop('materials');
    }
}
```

Listing 12.5: Materials Migration

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;


/*possible set workaround ->  https://coderwall.com/p/mo1gew/custom-datatype-in-laravel-schema-
    builder*/

class CookieBases extends Migration {

    /**
    * Run the migrations.
    *
    * @return void
    */
    public function up() {
        Schema::create('cookie_bases', function (Blueprint $table) {
            $table->increments('id');
            $table->string('cookie_name');
            $table->string('cookie_shapeCategory');
            $table->unsignedInteger('cookie_shape_id');
            $table->text('allergens');
            $table->integer('price');
            // $table->text('ingredients');
            $table->double('weight', 5,2); // xxxx,xx
            $table->double('height', 5,2);
            $table->timestamps();

            $table->unique(['cookie_name','cookie_shapeCategory']);
        });

        Schema::table('cookie_bases', function($table) {
            $table->foreign('cookie_shape_id')->references('id')->on('cookie_shapes');
        });
    }

    /**
    * Reverse the migrations.
    *
    * @return void
    */
    public function down() {
        Schema::drop('cookie_bases');
    }
}
```

Listing 12.6: Cookie-Bases Migration

```
<?php

```

```
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class TemplateSvgs extends Migration {
    /**
    * Run the migrations.
    *
    * @return void
    */
    public function up() {
        Schema::create('template_svgs', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->longtext('svg');
        });
    }
    /**
    * Reverse the migrations.
    *
    * @return void
    */
    public function down() {
        Schema::drop('template_svgs');
    }
}
```

Listing 12.7: Template-Svgs Migration

```
<?php
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
class CreateOrderFroms extends Migration {
    /**
    * Run the migrations.
    *
    * @return void
    */
    public function up() {
        Schema::create('order_forms', function (Blueprint $table) {
            $table->increments('id');

            $table->unsignedInteger('user_id');
            $table->unsignedInteger('cookie_id');
            $table->integer('cookie_amount');

            $table->longtext('decoration');

            $table->longtext('decorationOnCookie');
            $table->string('bill_street');
            $table->integer('bill_streetNr');
            $table->string('bill_country');
            $table->string('bill_state');
            $table->string('bill_city');
            $table->integer('bill_zip');

            $table->string('deliver_street');
            $table->integer('deliver_streetNr');
```

```
31              $table->string('deliver_country');
                $table->string('deliver_state');
33              $table->string('deliver_city');
                $table->integer('deliver_zip');
35
                $table->timestamp('converted_on')->nullable();
37              $table->timestamp('executionQueue_added_on')->nullable();
                $table->timestamp('execution_finished_on')->nullable();
39              $table->timestamps();
            });
41
            Schema::table('order_forms', function($table) {
43              $table->foreign('cookie_id')->references('id')->on('cookie_bases');
                $table->foreign('user_id')->references('id')->on('users');
45          });
        }
47      /**
        * Reverse the migrations.
49      *
        * @return void
51      */
        public function down() {
53          Schema::drop('order_forms');
        }
55  }
```

Listing 12.8: Order-Forms Migration

Abbildung 12.1: AllDrawingSVGElementHandler für Extremwert-Suche (Klassendiagramm)

Zeitaufzeichnung: Miko Christoph

Start	Bezeichnung	Dauer
Sprint 1		0,0
18.09.2016	Basiskonfiguration Server (Apache2, MySQL, WordpPressBlog	1,0
21.09.2016	Evaluierungen (ITP-Unterricht)	3,5
26.09.2016	Eignigung auf Website Schema Sportort und Benennung für Kundeneigene Verzierungen PostgreSQL vs MySQL	3,3
26.09.2016	Erstellung der Datenbank Anlegen des Users 'laravel' Konfigurieren der user privileges des Users 'laravel'	0,7
28.09.2016	Vorbereitung der Sprintabnahme + Abnahme durch List	5,2
Sprint 2		0,0
03.10.2016	Evaluierung	4,0
05.10.2016	Evaluierung	3,5
12.10.2016	Evaluierung	3,5
17.10.2016	Bugfix: Laravel Seeds and Migartions	4,3
19.10.2016	Evaluierung	3,5
24.10.2016	login / bestellformular backend	2,8
28.10.2016	Sprintabnahme	0,8
Sprint 3		0,0
03.11.2016	Retrospektive	3,5
07.11.2016	Umsetztung Retrospektive; z.B.: Erstellen eines Interfaces für das Bestellformular	2,0
09.11.2016	orderfrom backend +database table	3,2
09.11.2016	testing backend	2,0
10.11.2016	bugfixing token error	0,2
10.11.2016	fixing session laravel error -> solution: use new laravel installation TODO: merge with master / jasmin	2,8
21.11.2016	acc mng und login interface	0,9
22.11.2016	backend for user management	1,7
22.11.2016	added testing possibility for backend user management + email is now changeable	0,2
23.11.2016	Doku Sprintabnehme TdoT Teil 1	1,2
23.11.2016	Doku Sprintabnehme TdoT Teil 2 + sachliche Aufklärung Miriam how easy is laravel for real	0,2
23.11.2016	Vorbereitung Dokument sprintabnahme + Sprintabnahme + Problem Jasmin angesprochen	3,3
Sprint 4		0,0
28.11.2016	sprinteinteilung / userstory überarbeiten	0,9
30.11.2016	fixen laravel login rebasing maszer branch and resolven merge conflicts (unfinished)	0,8
30.11.2016	laracast tutorials + jasmin talk + evaluation java lib for csv	5,7
21.12.2016	abklärung mit roman bezüglich svg verarbeitung	1,5

Zeitaufzeichnung: Miko Christoph

27.12.2016	Evaluierung Teil 1vektoranzahl finden (php or java) - unfinished	3,0
28.12.2016	Evaluierung Teil 2 vektoranzahl finden (php or java) - unfinished	2,5
28.12.2016	bugfixing Backend userlogin (remember me)	0,5
29.12.2016	Vektor anzahl finden via php svg lib	1,6
01.01.2017	debugging - vektor anzahl finden	3,0
04.01.2017	evaluieren anderer methoden für vektor anzahl	1,5
06.02.2017	Back- & Frontend Bestellformular	7,0
07.02.2017	Back- & Frontend Bestellformular	7,0
08.02.2017	Back- & Frontend Bestellformular	7,0
09.02.2017	Back- & Frontend Bestellformular	7,0
09.02.2017	Stripe implementierung Back- & Frontend	3,0
10.02.2017	Back- & Frontend Bestellformular	2,0
11.02.2017	Database rework and cleanup	3,5
14.02.2017	Evaluierung Laravel exlir und vue2	2,0
15.02.2017	Teambesprechung und Server Sicherheits Evaluierung	4,7
22.02.2017	Bugfixing StepOne StepTwo Frontend	3,7
23.02.2017	Accountmanagement fertigstellung (Backend + Frontend)	3,7
24.02.2017	Weiterarbeiten StepThree + Bugfixing Javascript	5,0
25.02.2017	Rework Laravel Routes Backend	3,0
26.02.2017	Evaluierung Robotersicher + Notes	5,0
01.03.2017	Bugfixen Bestellformular Frontend + Backend	3,7
02.03.2017	Redesign StepFour + StepFive (Frontend)	4,0
12.03.2017	allgemeines Bugfixes Frontend (Javascript) + Backend	9,0
13.03.2017	DiplArbeit schreiben+ Feedback einarbeiten	6,0
14.03.2017	DiplArbeit schreiben + fehler ausbessern + minor bugfixing frontend + backend	10,0
15.03.2017	DiplArbeit weiterschreiben	4,5
16.03.2017	DiplArbeit "fertigstellen"	11,0
22.03.2017	Da schreiben / mergen / list besprechung	7,5
22.03.2017	Feedback einarbeiten	5,0
25.03.2017	DA "fertigstellen"	5,0
27.03.2017	Da finalisieren / Drucken	13,0

Summe: 220,4

Zeitaufzeichnung: Gschiegl Roman

Start	Bezeichnung	Dauer
Sprint 1		0,0
12.09.2016	Arbeit am Projektantrag + Konzeptentwicklung	5,0
13.09.2016	Arbeit am Projektantrag + Konzeptentwicklung	5,0
14.09.2016	Arbeit am Projektantrag + Konzeptentwicklung	5,0
15.09.2016	Arbeit am Projektantrag + Konzeptentwicklung	5,0
16.09.2016	Arbeit am Projektantrag + Konzeptentwicklung	4,0
Sprint 2		0,0
11.10.2016	Machbarkeit: Rohupload der Verzierung aufbereiten	2,0
12.10.2016	Machbarkeit: Verzierung prototypisch umsetzen für Materialsanalyse	4,3
16.10.2016	Entwurf der Idee & grobes Mockup fuer ein "Operator-Interface"	0,0
19.10.2016	Verzierungsprototyp aus LEGO gebaut (Halterung+Motorisierung)	4,3
20.10.2016	Verzierungsprototyp aus LEGO gebaut (Halterung+Motorisierung)	2,7
21.10.2016	Verzierungsprototyp aus LEGO verbessert	1,3
23.10.2016	Recherche Roboter dynamisch ansteuern	1,5
24.10.2016	Support anderer Teammitglieder	0,2
24.10.2016	JOpenShowVar implementieren (testweise)	5,5
25.10.2016	JOpenShowVar Bugfixing	3,5
25.10.2016	Dokumentation JOpenShowVar und Alternativen	1,0
Sprint 3		0,0
01.11.2016	KRL-Recherche: Geschwindigkeit mit Variable angeben	1,2
04.11.2016	Beispielmotiv-Integration: Von Java ueber KRL bis zum Roboter	2,8
04.11.2016	Beispielmotiv-Integration: Von Java ueber KRL bis zum Roboter	2,8
07.11.2016	Bugfixing: Integration von Java zu Roboter	2,7
08.11.2016	Bugfixing: Integration von Java zu Roboter	2,3
08.11.2016	Bugfixing: Integration von Java zu Roboter	2,3
09.11.2016	Programmierung der Ampel (Anzeige Produktionsstatus)	2,0
22.11.2016	Support von Jasmin (Bestellformular Fehlerausgabe AJAX + Bugfix)	2,7
Sprint 4		0,0
28.11.2016	Sprintplanung	1,0
28.11.2016	Researched for Splines and Curves on the KUKA robot	2,8
30.11.2016	Researched for Splines and Curves on the KUKA robot	3,2
06.12.2016	Researched for Splines and Curves on the KUKA robot	2,2

Zeitaufzeichnung: Gschiegl Roman

06.12.2016 Userstories umdefiniert bzw. erweitert, auch neue erstellt	0,8
07.12.2016 An Preprocessor I gearbeitet: generelle Struktur und Polygon-Übersetzung	5,7
08.12.2016 Preprocessor I: Neustrukturierung und XML-Outputstring	2,7
09.12.2016 Preprocessor I: Verschiedene Converter fuer SVG-Objekte geschrieben	2,0
09.12.2016 Preprocessor I: Neustrukturierung PolygonConverter und PolylineConverter geschrieben	0,6
11.12.2016 Preprocessor I: Recherche und Konzept für's <g> Element	1,0
12.12.2016 Preprocessor I: De Casteljau für <path> recherchiert und getestet	3,3
13.12.2016 Preprocessor I: teilweise Umsetzung <path> Konverter	4,5
17.12.2016 Preprocessor I: An quadratischen Kurven gearbeitet + Darstellung KurvenannÃ¤herung	1,0
20.12.2016 Preprocessor I: Relative Parameter zu absoluten + Pencil position refactored	2,0
20.12.2016 Preprocessor I: Architektur f. Multi-Shape Converter definiert und umgesetzt & fuer PathConverter umgesetzt	1,5
21.12.2016 Preprocessor I: Support fuer lines in <path> entwickelt, ausserdem smooth cubic curves in <path>	3,0
23.12.2016 Preprocessor I: Smooth Quadratic Curves, small-letter moveto, bugfix	2,0
23.12.2016 Preprocessor I: Support fuer 'close-path' cmd + Bugfix	2,5
27.12.2016 Preprocessor Stage0: an Rescale Verzierungsgrafik gearbeitet	4,0
28.12.2016 Preprocessor Stage0: an Rescale Verzierungsgrafik gearbeitet	2,3
28.12.2016 Preprocessor Stage0: an Rescale Verzierungsgrafik gearbeitet	5,0
30.12.2016 Preprocessor I: an Konzept fuer <text> gearbeitet	4,0
31.12.2016 Preprocessor I: an Konzept fuer <ellipse> und elliptial arc in <path> gearbeitet	2,5
03.01.2017 Preprocessor I: an Konzept fuer elliptial arc in <path> gearbeitet	1,8
04.01.2017 Preprocessor I: an Umsetzung Konvertierung fuer elliptical arc in <path> gearbeitet	2,0
05.01.2017 Dokumentation Javaklassen, Implementierung der W3C SVG-Empfehlungen fuer Konverter, Git-Repository Management	2,5
06.01.2017 Dokumentieren fuer Sprint 4	3,0
07.01.2017 Dokumentieren fuer Sprint 4	2,0
07.01.2017 Dokumentieren fuer Sprint 4	2,0
08.01.2017 Bugfixing SVG Preprocessing I	1,5
10.01.2017 Ellipsen-Konvertierung bearbeitet	1,2
12.01.2017 Uebersetzung von <text> Elementen	3,5
13.01.2017 Uebersetzung von <text> Elementen: weitergearbeitet in Java und Erweiterungen im Python-Script	3,5
14.01.2017 Theoretische Ueberlegung zu weiterer <text>-Uebersetzung	0,5
15.01.2017 Verschieben der einzelnen Buchstaben im <text> zu Koordinatenursprung (python script)	2,0
16.01.2017 TODOs des TextConverter abgearbeitet	5,0
17.01.2017 TextConverter: Dokumentation + Support fuer mehrere Wörter im <text>	5,2

Zeitaufzeichnung: Gschiegl Roman

22.01.2017 Vorbereitung Gespraech Prof. Guentoer	0,5
23.01.2017 KRL-Bugfix Endbewegung nach Verzierungsauftrags	5,3
24.01.2017 Arbeit an automatisierter Ausführung von Roboterprogrammen	5,0
24.01.2017 KRL-Bugfix Endbewegung nach Verzierungsauftrags	4,0
25.01.2017 Arbeit an automatisierter Ausführung von Roboterprogrammen	3,0
26.01.2017 Vorbereitung Tag der offenen Tür	5,0
05.02.2017 Arbeit am Bestellvorgang + Stage 0 Preprocessor	7,0
06.02.2017 Arbeit am Bestellvorgang + Stage 0 Preprocessor	7,0
07.02.2017 Arbeit am Bestellvorgang + Stage 0 Preprocessor	7,0
08.02.2017 Arbeit am Bestellvorgang + Stage 0 Preprocessor	7,0
09.02.2017 Arbeit am Bestellvorgang + Stage 0 Preprocessor	7,0
15.02.2017 Bestellvorgang Frontend Javascripting	4,0
18.02.2017 Bugfixing <path> repositioning + <svg> clipping preparation	3,0
19.02.2017 Implemented Stage0:Filtering + Stage0:Ungrouping	5,0
20.02.2017 Bugfixing various Stage0 modules	1,0
20.02.2017 Webshop: Worked on "Use template for cookie decoration"	4,0
21.02.2017 Webshop: Aufruf Preprocessor für kundeneigene Keksdesigns	0,8
21.02.2017 Webshop weiterentwickelt	5,3
22.02.2017 Webshop Step3-Entwicklung fortgesetzt	1,0
22.02.2017 Vorbereitung Präsentation zu Sprintende	3,0
23.02.2017 Bugfix Roboterkommunikation Zirkularbewegung	1,2
28.02.2017 Preprocessor-Weiterentwicklung: nur Teile der vollen Funktion wünschbar (Selektion)	2,5
28.02.2017 Webshop: Aufruf Preprocessor (only Rescaling) & Preprocessor-Weiterentwicklung <svg> "viewBox" fuer globale Weite+Hoehe	2,0
01.03.2017 Errorhandling + Error-Reporting global	2,0
02.03.2017 Diverse Verbesserungen + Neues Exception-Design fuer Fehlerfaelle und Ausnahmefaelle + Logging per Java-Utils	7,0
03.03.2017 Umsetzung Keksobot-Exceptions und Weiterleitung an Benutzerschnittstelle	3,0
05.03.2017 SVG-Validierung fuer Preprocessor (Stage 0) + Fehlerausgaben	2,0
05.03.2017 IP/Port in Keksobot.properties konfigurierbar	1,0
08.03.2017 Rücksichtnahme der Auftragsgröße (alle Keksobot-Komponenten)	5,0
10.03.2017 Roboterkommunikation Fehlerfall "Roboter nicht erreichbar" + Tolerierbarkeit in Benutzerschnittstelle	1,5
11.03.2017 automatisches Installierungs-Script fuer Keksobot-Komponenten	1,3
11.03.2017 System-D Servicedateien fuer Keksobot-Komponenten entwickelt + Testing in virtualisierter Umgebung	3,0
13.03.2017 Dokumentation Preprocessor beendet	1,5

Zeitaufzeichnung: Gschiegl Roman

15.03.2017 Einleitung geschrieben "Order Converter" + Einbindung vorhandene Doku "Order Converter" (Umwandlung in anlagenneutr. Format)	2,0
Summe:	284,1

Zeitaufzeichnung: Hauer Miriam

Start	Bezeichnung	Dauer
Sprint 1		0,0
19.09.2016	Evaluierung Algorithmus	1,5
21.09.2016	Evaluierung, Latex Einführung	3,5
24.09.2016	Evaluierung, Schema, Website, etc	3,1
25.09.2016	Evaluierung	4,0
25.09.2016	Evaluierung	1,7
28.09.2016	Evaluierungssprintabnahme. XML besprechnung, mergen	5,2
Sprint 2		0,0
03.10.2016	Grundriss Algorithmus	7,2
03.10.2016	Kuka Kommunikation	5,0
05.10.2016	XML to Kuka Programmierung	4,7
10.10.2016	Kuka Kommunikation	3,0
12.10.2016	XML to Kuka Programmierung	4,7
19.10.2016	Quetschmodul basteln	5,2
20.10.2016	Evaluierung quetschmodul	6,7
24.10.2016	Kuka Kommunikation	5,5
25.10.2016	Kuka Kommunikation	6,0
26.10.2016	Algorrithm	6,0
28.10.2016	Sprint abnahme	1,7
Sprint 3		0,0
03.11.2016	Retrospektive	4,0
04.11.2016	Kuka Kommunikation bug fixing und erweiterungen	6,5
07.11.2016	Gruppengespräch, Kuka Kommunikation	6,0
09.11.2016	Verkabelung der Lampen im Schaltschrank, Programm Erweiterung zwecks lampen, 1/2 Knöpfen verkabelt	6,5
21.11.2016	#16 Server : Konfiguration und Netzwerk	0,0
21.11.2016	allgemeine besprechung, dokumentation	5,9
23.11.2016	korrektur lesen	0,8
23.11.2016	sprint abnahme, mergen	4,3
Sprint 4		0,0
28.11.2016	Sprint besprechhung	0,9
28.11.2016	Evaluierung reinigung kuka	1,9
30.11.2016	Evaluierung reinigung, variablen definieren	4,4

Zeitaufzeichnung: Hauer Miriam

06.12.2016 besprechung	1,0
07.12.2016 Sprint arbeit reinigung	3,7
14.12.2016 Sprint arbeit reinigung	3,7
21.12.2016 Brett bauen, Erweiterung testen	4,2
04.01.2017 sprint doku	2,5
11.01.2017 präsentationsbesprechung	0,9
18.01.2017 glasurspender	7,7
24.01.2017 allgemeine besprechung+doku	3,7
25.01.2017 allgemeine besprechung+doku	3,7
26.01.2017 Verkabelung im Kasten und anpassung proeprocessor 2 für externen aufruf	8,5
26.01.2017 Kukaschrank verkabeln	7,5
27.01.2017 Tag der offenen Tür + fixes	6,0
06.02.2017 php sockets + text for website	7,0
07.02.2017 worked on blog entry and texts	7,0
08.02.2017 worked on colors and order datatype	7,0
09.02.2017 worked on blog entry, texts, documentation and XMLtoList	7,0
15.02.2017 schreiben projektmanagement teil für DA, seeds machen	6,5
28.02.2017 XMLtoList, dokumentation	5,0
01.03.2017 besprechung mit prof list, dokumentieren, website colors	6,7
03.03.2017 colors for webshop + javascript functions	4,7
13.03.2017 diplomarbeit schreiben	3,0
14.03.2017 diplomarbeit schreiben	9,0
15.03.2017 diplomarbeit korrektur lesen + ablaufdiagrame + fotos einfügen	6,5
16.03.2017 diplomarbeit korrektur lesen	7,8
Summe:	245,8

Literatur

[1] aauc-mechlab und Github contributers. *Github-Repo von JOpenShowVar*. abg. am 23.10.2016. URL: https://github.com/aauc-mechlab/JOpenShowVar.

[2] ajtrichards. *Restricting MySQL connections from localhost to improve security*. 22.03.2017. URL: http://stackoverflow.com/questions/13208614/restricting-mysql-connections-from-localhost-to-improve-security.

[3] Avi Alkaly. *Designing Integrated High Quality Linux Applications*. abg. am 12.03.2017. URL: http://www.tldp.org/HOWTO/HighQuality-Apps-HOWTO/fhs.html.

[4] *Allgemeine Informationen (Sensible Daten)*. abg. am 27.03.2017. URL: https://www.help.gv.at/Portal.Node/hlpd/public/content/244/Seite.2440300.html.

[5] *Composer*. abg. am 27.03.2017. URL: https://getcomposer.org/.

[6] *Create User Syntax*. 22.03.2017. URL: https://dev.mysql.com/doc/refman/5.7/en/create-user.html.

[7] *DIY Lego Mindstorms EV3 custom touch sensor*. 14.03.2017. URL: https://www.youtube.com/watch?v=3rIK1XeKHEU.

[8] *Eloquent*. abg. am 27.03.2017. URL: https://laravel.com/docs/5.4/eloquent.

[9] *Erkennung von Kollisionen bei Robotergliedmaßen anhand sensometrischen Daten*. URL: http://www.neurorobotik.de/downloads/publications/2011%20Bethge%20-%20Kollisionserkennung%20anhand%20von%20sensomotorischen%20Daten.pdf.

[10] *Finding a Point on a Bézier Curve: De Casteljau's Algorithm*. abg. am 08.01.2017. URL: https://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/de-casteljau.html.

[11] Robert Grycke. *Klötzchen für Kekse wie du mit Lego deine Plätzchen dekorierst*. 14.03.2017. URL: http://blog.vodafone.de/gadgets-und-wearables/kloetzchen-fuer-kekse-wie-du-mit-lego-deine-plaetzchen-dekorierst/.

[12] *handbuch agilus sixx*. 22.03.2017. URL: https://github.com/mhauer-tgm/Syt/blob/master/Spez_KR_AGILUS_sixx_de.pdf.

[13] *How to Disable Password Authentication for SSH*. abg. am 27.03.2017. URL: http://support.hostgator.com/articles/specialized-help/technical/how-to-disable-password-authentication-for-ssh.

[14] *HTTPD Apache*. abg. am 27.03.2017. URL: https://httpd.apache.org/.

[15] *Hypertext Transfer Protocol Secure*. 22.03.2017. URL: https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol_Secure.

[16] Adobe® Systems Incorporated. *PDF Reference sixth edition*. abg. am 31.01.2017. URL: https://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf.

[17] Adobe Systems Incorporated. *PostScript language reference manual - 3rd ed.* abg. am 20.03.2017. URL: https://www.adobe.com/products/postscript/pdfs/PLRM.pdf.

[18] *Input Devices*. 14.03.2017. URL: http://docs.ev3dev.org/projects/lego-linux-drivers/en/ev3dev-jessie/sensors.html.

[19] *JOpenShowVar*. 20.03.2017. URL: https://github.com/aauc-mechlab/JOpenShowVar.

[20] *Kapazitiver Sensor*. abg. am 16.03.2017. URL: https://de.wikipedia.org/wiki/Kapazitiver_Sensor.

[21] *KUKA.Ethernet KRL 2.1, For KUKA System Software 8.2*. Issued: 21.03.2012. KUKA Roboter GmbH.

[22] *Laravel Blade*. abg. am 27.03.2017. URL: https://laravel.com/docs/5.4/blade.

[23] *Laravel Columns*. abg. am 27.03.2017. URL: https://laravel.com/docs/5.4/migrations#columns.

[24] *Laravel Framework*. abg. am 27.03.2017. URL: https://laravel.com/.

[25] *Lets encrypt*. abg. am 27.03.2017. URL: https://letsencrypt.org/.

[26] Shaun Lewis. *How To Setup a Firewall with UFW on an Ubuntu and Debian Cloud Server*. 22.03.2017. URL: https://www.digitalocean.com/community/tutorials/how-to-setup-a-firewall-with-ufw-on-an-ubuntu-and-debian-cloud-server.

[27] *Lichtschranken*. abg. am 16.03.2017. URL: https://de.wikipedia.org/wiki/Lichtschranke#Lichtgitter.

[28] Alex Lockwood. *ArcToBezier.java*. abg. am 27.03.2017. URL: https://gist.github.com/alexjlockwood/8ca9228e861222866c666513ed401a71#file-arctobezier-java.

[29] L. Maisonobe. *Drawing an elliptical arc using polylines, quadratic or cubic Bezier curves*. abg. am 15.02.2017. URL: https://www.spaceroots.org/documents/ellipse/elliptical-arc.pdf.

[30] *Model View Controller*. abg. am 27.03.2017. URL: https://de.wikipedia.org/wiki/Model_View_Controller.

[31] *MySQL*. abg. am 27.03.2017. URL: https://www.mysql.de/.

[32] Neofpo. *SSH with authentication key instead of password*. abg. am 27.03.2017. URL: https://debian-administration.org/article/530/SSH_with_authentication_key_instead_of_password.

[33] *NginX*. abg. am 27.03.2017. URL: https://www.nginx.com/resources/wiki/.

[34] *optokoppler*. 22.03.2017. URL: https://de.wikipedia.org/wiki/Optokoppler.

[35] *PHP*. 22.03.2017. URL: https://wiki.ubuntuusers.de/PHP/.

[36] *PostgreSQL*. abg. am 27.03.2017. URL: https://www.postgresql.org/.

[37] *Protokolle*. 22.03.2017. URL: https://github.com/mhauer-tgm/Syt/tree/master/Roboterprotokolle4Jahrga

[38] *Richtlinie 2006/42/EG*. 22.03.2017. URL: http://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32006L0042.

[39] *SSH Key - The Forgotten Access Credential*. 22.03.2017. URL: https://www.ssh.com/ssh/key/.

[40] Stripe. abg. am 27.03.2017. URL: https://stripe.com/at.

[41] Stripe. *Stripe-PHP*. abg. am 27.03.2017. URL: https://packagist.org/packages/stripe/stripe-php.

[42] *tactile sensor*. abg. am 16.03.2017. URL: https://en.wikipedia.org/wiki/Tactile_sensor.

[43] Chris Taylor. abg. am 08.01.2017. URL: https://math.stackexchange.com/questions/43947/casteljaus-algorithm-practical-example#43968.

[44] *TrueType™ Reference Manual*. abg. am 16.03.2017. URL: https://developer.apple.com/fonts/TrueType-Reference-Manual/.

[45] Hazel Virdó. *How To Install MySQL on Ubuntu 16.04*. abg. am 27.03.2017. URL: https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-16-04.

[46] World Wide Web Consortium (W3C). *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. abg. am 07.01.2017. URL: https://www.w3.org/TR/SVG/Overview.html.

[47] Wikipedia. *HTTP-Anfragemethoden*. 22.03.2017. URL: https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol#HTTP-Anfragemethoden.

[48] *Work Visual*. 14.03.2017. URL: https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/software/systemsoftware/kuka_systemsoftware/kuka_work-visual.

[49] *WorkVisualAnleitung*. 22.03.2017. URL: https://github.com/mhauer-tgm/Syt/blob/master/KST_WorkVisual_en.pdf.

[50] *XSS (Cross Site Scripting) Prevention Cheat Sheet*. abg. am 27.03.2017. URL: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.

Abbildungsverzeichnis