Fakultät Informatik
Hochschule Furtwangen
Robert-Gerwig-Platz 1
D-78120 Furtwangen

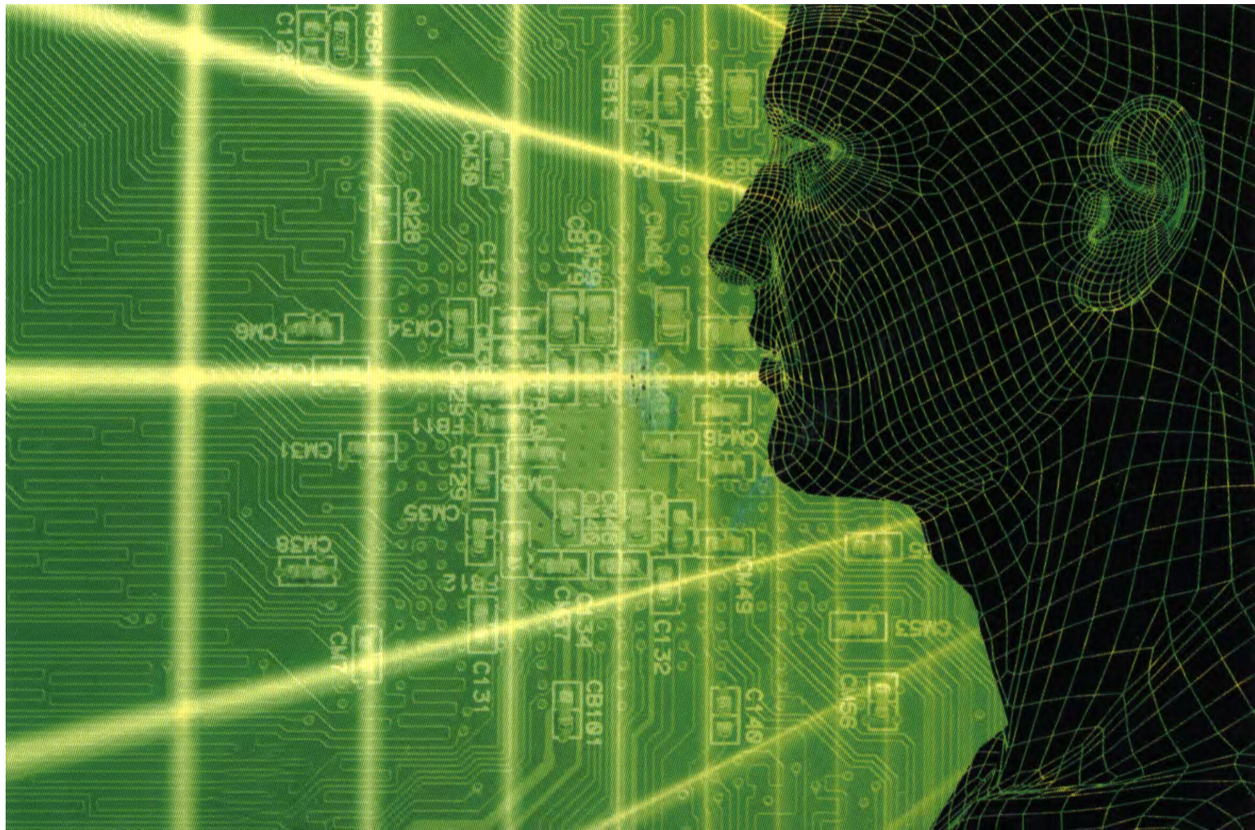
ISBN 978-3-00-051538-5

informatikJournal 2016/17

HOCHSCHULE FURTWANGEN UNIVERSITY HFU

informatikJournal 2016/17

Aktuelle Berichte aus Forschung und Lehre
der Fakultät Informatik



informatikJournal 2016/17

Aktuelle Berichte aus Forschung und Lehre
der Fakultät Informatik

Herausgeber

Fakultät Informatik
Hochschule Furtwangen
Robert-Gerwig-Platz 1
D-78120 Furtwangen

http://www.hs-furtwangen.de/studierende/fakultaeten/informatik.html

Redaktion

Prof. Dr. Steffen Thiel

© 2016 Fakultät Informatik, Hochschule Furtwangen
Alle Rechte vorbehalten

Printed in Germany

ISBN 978-3-00-051538-5

9 783000 515385 >

Inhaltsverzeichnis

Vorwort v

Assistive Systeme 1

Kamerabasierte Assistenz für Blinde und Sehbehinderte 3
Judith Jakob, Elmar Cochlovius und Christoph Reich

Autonome Roboterschwärme in geschlossenen Räumen 11
Roberto Ciano und Dimitrij Klesev

Softwaretechnik 17

Beitrag der Java Modeling Language zur Qualität 19
Nicole Pfeffermann und Christine Rotfuß

Einfluss von aspektorientierter Programmierung auf die Laufzeit 27
Aiko Isselhard und Nico Konop

Verwendung von aspektorientierter Programmierung zur Vereinfachung von entfernten Methodenaufrufen mit Apache Thrift 35
Stefan Retzlaff und Serbest Hammade

Event Processing und Verteilte Systeme 41

Event Processing Networks 43
Martin Haschke und Stefan Retzlaff

Denial of Service Angriffe auf SOAP Web Services 49
Martin Haschke, Stefan Retzlaff und Friedbert Kaspar

Anwendung von linearen Klassifizierern zur Kategorisierung von Texten 55
Martin Haschke und Aiko Isselhard

Erkennung und Synchronisation von Multimedia-Dateien 61
Emil Gabert

Vorwort

Liebe Leserinnen und Leser,

herzlich willkommen zu einer neuen Ausgabe des *informatikJournal*, der Fachpublikation der Fakultät Informatik. In dieser Ausgabe haben wir wieder ausgewählte Arbeiten aus den vielfältigen Forschungsaktivitäten und Seminaren unserer Fakultät zusammengestellt.

Die Beiträge der vorliegenden Edition 2016/17 beschäftigen sich mit aktuellen Fragestellungen der Informatik und umfassen u. a. die Themengebiete Assistive Systeme, Softwaretechnik sowie Event Processing und Verteilte Systeme. Diese Gebiete haben einen hohen Stellenwert für die Ausbildung von Informatikerinnen und Informatikern. Sie bieten schon heute interessante Lösungsansätze zur Verbesserung der industriellen Softwareentwicklung.

Wir danken unseren Doktoranden, Masterstudenten und Absolventen für das Engagement und die zahlreichen Beiträge in dieser Ausgabe. Ferner danken wir allen beteiligten Kolleginnen und Kollegen für die freundliche Unterstützung und kompetente Betreuung der Arbeiten.

Wir wünschen Ihnen eine interessante Lektüre und freuen uns über Ihr Feedback.

Furtwangen, Februar 2016

Prof. Dr. Mohsen Rezagholi
(Dekan)

Prof. Dr. Steffen Thiel
(Redaktion)

Assistive Systeme

Kamerabasierte Assistenz für Blinde und Sehbehinderte

State of the art

Judith Jakob, Prof. Dr. Elmar Cochlovius, Prof. Dr. Christoph Reich
Fakultät Informatik
Hochschule Furtwangen University
78120 Furtwangen, Deutschland
{Judith.Jakob, Elmar.Cochlovius, Christoph.Reich}@hs-furtwangen.de

***Abstract*—Der Einsatz von Kameras im Zusammenspiel mit intelligenten Bildverarbeitungs- und erkennungsalgorithmen ist ein weit verbreiteter Ansatz zur Unterstützung von blinden und sehbehinderten Menschen im Alltag. Zunächst wird der allgemeine Systemaufbau von kamerabasierten Smartphone-Apps und mobilen Assitenzsystemen für Blinde und Sehbehinderte vorgestellt. Diese werden sortiert nach den Einsatzgebieten Vorlesen von Text, Erkennen von Gesichtern und Objekten, Wahrnehmung der Umgebung sowie Navigation und Hinderniserkennung beschrieben und anhand der Kriterien Systemaufbau, Kosten und Verfügbarkeit, Bedienbarkeit sowie Funktionalität und Performanz miteinander verglichen. Ferner wird ein geplantes mobiles Assistenzsystem bestehend aus einer Kamera, einem Smartphone und einer Cloud-Anwendung vorgestellt**

Keywords—Bildverarbeitung; Kamera; Blinde und Sehbehinderte; Mobile Assistenzsysteme

I. EINLEITUNG

Der Einsatz von Kameras im Zusammenspiel mit intelligenten Bildverarbeitungs- und erkennungsalgorithmen ist ein weit verbreiteter Ansatz zur Unterstützung von blinden und sehbehinderten Menschen im Alltag. Das Spektrum reicht dabei von einer geeigneten Darstellung des Bildmaterials über die Interpretation von Bildinhalten in bestimmten Situationen bis hin zu Systemen, mit denen signifikante Anteile der Umgebung des Anwenders diesem über nichtvisuelle Sinneswahrnehmungen zugänglich gemacht werden. Dabei kommen unterschiedliche Technologien zum Einsatz: von der Smartphone-App bis hin zu für eine bestimmte Anwendung entwickelten mobilen Assistenzsystemen.

Im Kontext der Forschungsaktivitäten im Bereich Smart Home und Cloud Computing an der Fakultät Informatik werden Ansätze zur kamerabasierten Hilfestellung für blinde und sehbehinderte Menschen untersucht. In Gesprächen mit der Interessensvertretung der Anwender, dem Blinden- und Sehbehindertenverein Südbaden [47], wird deutlich, dass vor allem Bedarf an mobilen Assistenzsystemen für Blinde und Sehbehinderte besteht. Im Gegensatz zu stationären Systemen des häuslichen Umfelds, können mobile Systeme in zahlreichen Umgebungen zum Einsatz kommen und zu einer wertvollen Steigerung der eigenständigen Mobilität von Blinden und Sehbehinderten beitragen.

Dieser Beitrag gibt zunächst eine Übersicht über den Systemaufbau kamerabasierter, mobiler Assistenzsysteme. Danach werden vorhandene und in der Entwicklung befindliche Systeme, untergliedert nach ihrem Einsatzgebiet vorgestellt und auf die Referenzarchitektur abgebildet. Die in Kapitel III identifizierten Einsatzgebiete umfassen dabei das Vorlesen von Text, die Erkennung von Gesichtern und Objekten, die Wahrnehmung der Umgebung sowie Navigation und Hinderniserkennung. In Kapitel IV werden die vorgestellten Systeme nach den Kriterien Systemaufbau, Kosten und Verfügbarkeit, Bedienbarkeit sowie Funktionalität und Performanz analysiert und verglichen. Anschließend wird in Kapitel V ein Ausblick auf weitere Fragestellungen und mögliche zukünftige Forschungsarbeiten gegeben, bevor im abschließenden Kapteil VI eine Zusammenfassung erfolgt.

II. KAMERABASIERTE ASSISTENZSYSTEME

In diesem Abschnitt wird die grundlegende System-Architektur kamerabasierter Assistenzsysteme (KBAS) für Blinde und Sehbehinderte dargestellt.

Der allgemeine Aufbau von KBAS für Blinde und Sehbehinderte ist in Abb. 1 mit Hilfe abstrakter Funktionsmodule dargestellt. Die Funktionsmodule beschreiben dabei eine abgrenzbare Teil-Funktionalität des Gesamtsystems, ohne Annahmen über die Implementierung in Hard- oder Software zu treffen. Je nach Systemverfügbarkeit lassen sich bestimmte Bildverarbeitungsfunktionen entweder in der CPU oder auch sehr effizient in modernen GPGPUs (General Purpose Graphical Processing Units) implementieren. KBAS bestehen aus a) einer *Bilderfassungseinheit*, b) einer *Bildverarbeitungseinheit* sowie c) einer *Ausgabeeinheit*.

a) Die *Bilderfassungseinheit* enthält üblicherweise eine Videokamera bzw. einen Bildsensor. Damit wird eine vorliegende Szene, z. B. in Blickrichtung des Anwenders, erfasst. Die Capture- und Vorverarbeitungsstufe zerlegt den Datenstrom in einzelne Frames und führt eine Anpassung von Helligkeit, Kontrast, Farbraum und weiterer Parameter durch. Sind Bilderfassungs- und Bildverarbeitungseinheit voneinander

räumlich getrennt, so müssen die angepassten Bildsignale geeignet in die Bildverarbeitungseinheit übertragen werden (Transmission). Aufgrund der hohen Bandbreitenanforderungen roher Videodaten erfordert dies zunächst eine Videocodierung zur Kompression, z. B. mit Hilfe des H.264-Verfahrens, wobei hier eine niedrige Verzögerungszeit (low latency mode) im Vordergrund steht.

b) In der *Bildverarbeitungseinheit* werden nun ggf. die empfangenen Video-Daten decodiert und die einzelnen Original-Frames wiederhergestellt. Im Funktionsmodul „Feature-Extraction“ erfolgt eine Analyse des Bildinhalts eines einzelnen Frames. Dabei geht es um die Anwendung grundlegender Transformationen etwa zur Identifizierung von Kanten, zusammenhängenden Objekten oder Bereichen einheitlicher Farbe. Das Ergebnis wird an die nächste Stufe der Bildverarbeitungspipeline weitergereicht. Im Funktionsmodul „Evaluation“ wird eine erste Interpretation des vorliegenden Frame-Inhalts anhand seiner extrahierten Features vorgenommen. Durch Vergleich mit Vorgängerframes lassen sich hier Bewegungsvektoren von Objekten sowohl absolut im Weltkoordinatensystem, als auch relativ zueinander identifizieren und ihre jeweilige Geschwindigkeit bestimmen.

Je nach Anwendung werden im nächsten Schritt relevante Kontextinformationen, die über eine lokale Datenbank oder ein Netzwerk abgerufen werden, genutzt, um entweder mit Hilfe der Evalutation weitere Informationen zur Verfügung stellen zu können oder um auf der Evalutation aufbauende Hypothesen zu untermauern bzw. zu verwerfen.

c) In der *Ausgabeeinheit* wird nun aus den vorangegangenen Berechnungen und daraus entstehenden Hypothesen ein anwenderrelevantes Ergebnis generiert, indem eine geeignete und ggf. kontextabhängige Schlussfolgerung gezogen wird. Beispielsweise wird in einer Navigationsanwendung ein Richtungshinweis (Route Guidance) berechnet, der den Anwender zu dem erkannten Zebrastreifen hinführen soll. Die Kommunikation mit dem Anwender erfordert im letzten Schritt die Erzeugung einer multimodalen Ausgabe, die visuell auf dem Display in für Sehbehinderte angemessener Weise, hörbar über den Audiokanal oder auch taktil über alternative Ausgabegeräte (Force Feedback) erfolgen kann.

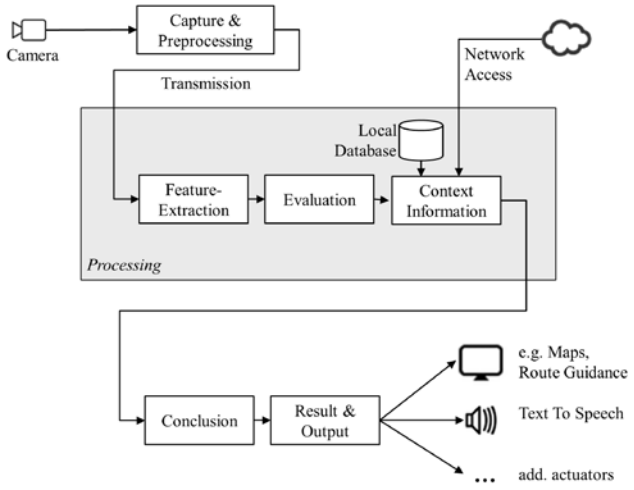


Abb. 1 High-Level-Architektur kamerabasierter Assistenzsysteme für Blinde und Sehbehinderte

III. Einsatzgebiete Kamerabasierter Assistenzsysteme

Nachfolgend werden die wichtigsten Einsatzgebiete kamerabasierter Assistenzsysteme beschrieben und diskutiert.

A. *Vorlesen von Text*

Da nicht alle Bücher und Texte in Brailleschrift verfügbar sind und gleichzeitig lediglich 27 Prozent der Sehgeschädigten in Deutschland die Brailleschrift beherrschen [1], stellt das Vorlesen von Text eine wichtige Hilfestellung für Blinde und Sehbehinderte dar.

Es existieren zahlreiche Apps, die ein aufgenommenes Bild analysieren und anschließend den darauf befindlichen Text per Text To Speech (TTS) ausgeben. Dies sind z. B. *KFNB-Reader* [24], *Blindsight Text Detective* [25] oder *ABBYY Textgrabber + Translator* [26].

Im Gegensatz dazu wurden und werden mobile Assistenzsysteme entwickelt, die in der Lage sind Text, der von einer laufenden Kamera aufgenommen wird, ohne Verzögerung vorzulesen. Dazu zählt das autarke und kommerziell erhältliche Assistenzsystem *OrCam* [2] der gleichnamigen Firma. Eine an einer Brille montierte Kamera erfasst die Szene im Gesichtsfeld des Anwenders. Er deutet in Richtung des zu erkennenden Textes, wodurch die TTS-Software eines mit der Kamera verbundenen Kleinrechners aktiviert wird und den betreffenden Text über eine Kopfhörereinheit ausgibt.

Der von MIT (Cambridge, MA, USA) entwickelte *FingerReader* [3] besteht aus einer kleinen Kamera, die ähnlich einem Ring, am Zeigefinger befestigt wird und mit einer Recheneinheit, einem PC, verbunden ist. Der Nutzer fährt mit dem Zeigefinger eine Textzeile entlang, die in synthetisierter Sprache wiedergegeben wird. Mit Hilfe von Motoren erhält der Nutzer taktile Rückmeldungen darüber, in welche Richtung der Finger bewegt werden muss, damit die Zeile von der Kamera korrekt erfasst und vom System weiterverarbeitet weren kann.

B. *Erkennen von Gesichtern und Objekten*

Mehrere Anwendungen und Assistenzsysteme ermöglichen es Sehgeschädigten zuvor abgespeicherte Gesichter und Objekte wiederzuerkennen. Die oben beschriebene *OrCam* etwa teilt dem Nutzer mit, ob sich im öffentlichen Raum ihm bekannte Personen befinden; mit der Objekterkennung kann beispielsweise im Supermarkt das gesuchte Produkt wiedergefunden werden. Weitere Erkennungsalgorithmen wie das Vorlesen der Nummer des sich nähernden Busses befinden sich derzeit in der Entwicklung.

Andere Anwendungen hingegen sind darauf ausgerichtet den Gesichtsausdruck und die Bewegungen eines Gesprächspartners zu analysieren und dem Nutzer mitzuteilen. Das System *Expression* [4] basiert auf Google Glass, während es sich bei *iFEPS* (improved Facial Impression Perception through Sound) [5] und *iMAPS* (interactive Mobile Affect Perception System) [6] um Android-Anwendungen mit einer Server-Verbindung zu einem PC handelt. *Expression* teilt dem Nutzer Kopfbewegungen (schaut nach rechts/links/oben/unten), Gesichtsausdrücke (z. B. Lachen) und Verhaltensweisen (Gähnen) des Gesprächspartners per TTS mit. Das

System *iMAPS* hingegen analysiert den Gesprächspartner gemäß des dimensionalen Ansatzes [27], der das Spektrum der menschlichen Empfindungen während einer Konversation über die drei Werte Valence (Wertigkeit), Arousal (Erregung) und Dominance (Dominanz) definiert. Die Art und Weise der Rückmeldung an den Nutzer wurde bislang nicht festgelegt. Hingegen baut *iFEPS* auf dem Facial Acting Coding System [28] auf, das Gesichtsausdrücke als Kombination von Bewegungseinheiten definiert. Die Bewegungen von Augenbrauen, Augen und Lippen werden in Ton kodiert und zum Nutzer übertragen.

Das Projekt *Listen2dRoom* [12] hat zum Ziel blinden und sehbehinderten Nutzern beim Verständnis des Layouts eines ihnen unbekannten Raums zu helfen, indem über eine tragbare Kamera bzw. die Smartphone-Kamera bestimmte Objekte im Raum, wie etwa Tische, Stühle und Mülleimer, erkannt und per Sprachausgabe mitgeteilt werden. Die Reihenfolge der Ausgabe lässt dabei Rückschlüsse auf die Anordnung im Raum zu.

Eine Vielzahl von Smartphone-Apps bietet Unterstützung beim Wiedererkennen von Objekten (z. B. *LookTel Recognizer* [7]) bzw. beim Erkennen von speziellen Objekten wie Geldscheinen (z. B. *LookTel Money Reader* [8]). Einen weitergehenden Ansatz verfolgt die App *TapTapSee* [9], die Bilder mit beliebigem Inhalt beschreibt. Allerdings ist es nach heutigem Stand der Bilderkennung nicht möglich, Bilder mit beliebigem Inhalt algorithmisch zu beschreiben. Daher setzt die App *TapTapSee*, wie in der Privacy Policy nachzulesen ist, neben einer algorithmischen Analyse auf Crowdsourcing.

C. Wahrnehmung der Umgebung

Während viele Anwendungen zur Unterstützung von Blinden und Sehbehinderten sich konkreten Problemstellungen widmen, existiert auch eine Vielzahl an Systemen, die einen umfassenden Ansatz verfolgen und die Umgebung als Ganzes für den Nutzer erfahrbar machen wollen. Meist werden dafür sogenannte Sensorical Substitution Devices (SSD) entwickelt, die Wahrnehmungen so umwandeln, dass sie über einen anderen als den beschädigten Sinn erfasst werden können.

Bereits 1992 schuf Meijer in [14] die Grundlagen für *vOICe*, ein SSD, das Kamerabilder in Audiosignale umwandelt. Die entsprechende Software ist heute kostenfrei für Windows und Android [15] verfügbar und kann auf Standardhardware, die mit einer Kamera ausgestattet ist, verwendet werden.

Das BMBF-Förderprogramm *OIWOB* – „Orientieren, Informieren, Warnen. Orientierungshilfe für Blinde“ [11] – hat das Ziel, Alltagssituationen eines blinden oder sehbehinderten Benutzers zu interpretieren und ihm die Orientierung zu erleichtern. Es soll ein autarkes System entwickelt werden, das mit Hilfe einer 3D-Tiefenkamera und Inertialsensorik die Umgebungsgeometrie erkennt. Ein dedizierter Rechner übernimmt die Extraktion der semantischen Information und erzeugt deren Interpretation und Sprachwiedergabe über eine Kopfhörereinheit.

Da der Hörsinn, gerade von blinden und sehbehinderten Menschen, zur Wahrnehmung der Umgebung von großer Bedeutung ist, konzentrieren sich andere Forschungen darauf, das Ohr zu entlasten und die optische Wahrnehmung statt auf den Hör- auf den Tastsinn zu übertragen. Eine Möglichkeit ist es, die Umgebung über die Zunge erfahrbar zu machen, da über diese aufgrund ihrer Empfindlichkeit eine differenzierte Wahrnehmung erreicht werden kann. Dafür werden Bilder mit einer Kamera aufgezeichnet, weiterverarbeitet und schließlich in elektronische Impulse umgesetzt, die an eine auf der Zunge befindliche Elektroden-Matrix gesendet werden. Die Grundlagen dafür werden erstmals in [30] vorgestellt; darauf aufbauende, aktuelle Forschungen finden sich etwa in [31] und [32]. Andere Forschungen übersetzen das bearbeitete Bildmaterial auf mit mehreren kleinen Motoren ausgestatte Gürtel (z. B. [33]), taktile Displays (z. B. [34]) oder übertragen das Signal direkt an den Fingerspitzen (z. B. [35]).

Statt die Umwelt über einen anderen als den Sehsinn erfahrbar zu machen, besteht die Möglichkeit die Darstellung von Objekten im Nahbereich mit Hilfe von Bildverbesserungsmaßnahmen zu optimieren und sie an die Restsehfähigkeit Sehbehinderter anzupassen. Die spezielle Sehhilfe *Smart Specs* (VA-ST, Oxford, UK) [10] erfasst die Szene über eine 3D-Tiefenkamera, ein Android-basierter Kleinrechner verstärkt Kontraste und filtert den Hintergrund, und zwei Mikroprojektoren blenden das verbesserte Bild über das natürliche Bild, das durch eine normale Brille hindurch erfasst wird.

D. Navigation und Hinderniserkennung

Im Bereich Navigation kommt als primäre Technologie GPS (siehe z. B. die Smartphone-Apps *Blindsquare* [16] und *LookTel Breadcrumb* [17]) zum Einsatz, für den Nahbereich ist GPS jedoch nicht geeignet. Neben den traditionellen Assistenzsystemen Langstock und Blindenhund, kommt zur Hinderniserkennung und Kollisionsvermeidung im Nahbereich häufig Ultraschall zum Einsatz [23]. Darüber hinaus leisten die in Abschnitt C beschriebenen Systeme zur Wahrnehmung der Umgebung ebenfalls einen wichtigen Beitrag in diesem Bereich. Mit der stetigen Weiterentwicklung der Bilderkennung und -verarbeitung werden vermehrt Kamerasysteme eingesetzt, um die GPS-basierte Navigation und Hinderniserkennung bzw. Kollisionsvermeidung zu präzisieren.

Bei dem Projekt *IMAGO* [18] geht es darum, die GPS-basierte Fußgänger-Navigation mit Hilfe von Bilderkennungsverfahren zu erweitern und zu präzisieren. Dabei werden aktuelle Bilder der Smartphone-Kamera des Anwenders mit zuvor aufgezeichneten Bildern derselben Route verglichen. Im Gegensatz dazu verwendet das Project *Crosswatch* [19] Satellitenbilder von Google Maps als Kontextinformation und vergleicht sie mit vom Nutzer gemachten 360-Grad-Panoramaaufnahmen zur genauen Lokalisierung des Nutzers.

Im Bereich Hinderniserkennung und Kollisionsvermeidung wird beispielsweise an Verbesserungen des meist schon vorhandenen Langstocks durch Ausstattung mit zusätzlichen Sensoren, etwa Infrarot und Ultraschall, aber auch Kameras, geforscht. Entsprechende Arbeiten, die unter anderem auf den Einsaz von Kameras setzen, finden sich z. B. in [36] und [37]. Weitere Forschungen konzentrieren sich darauf, den in der Ausbildung und Unterhaltung teuren und nicht für alle verfügbaren Blindenhund durch Roboter mit entsprechenden Fähigkeiten zu ersetzen. In [38] wird dabei eine handelsübliche

Smartphone-Kamera eingesetzt, um Hindernisse und Verkehrsinformationen erkennen und verarbeiten zu können.

Von den traditionellen Assistenzen unabhängige Kamerasysteme zur Hinderniserkennung können in drei Kategorien eingeteilt werden. Im ersten Fall werden Daten einer 2D-Kamera, oft einfache Smartphonekameras, meist mit Hilfe von Bewegungsschätzung durch Optical Flow verarbeitet (siehe [39], [40], [41]), im zweiten Fall handelt es sich um Daten von RGBD-Kameras, die neben den RGB-Farbwerten auch Tiefeninformationen zur Verfügung stellen (siehe [42], [43]) und im dritten Fall kommt Stereovision zum Einsatz (siehe [44], [45]).

Für Blinde und Sehbehinderte besonders wichtig ist die Detektion von nach unten führenden Treppen und von Objekten in Kopfhöhe, da diese mit dem Langstock im Allgemeinen nicht bzw. nur schwer zu erfassen sind. Die Arbeiten [42] und [44] widmen sich diesen Themen.

E. Sonstige Einsatzgebiete

Im Folgenden werden weitere Anwendungen und Forschungsgebiete vorgestellt, die sich nicht eindeutig einem der zuvor behandelten Einsatzgebiete zuordnen lassen.

Mit einem Barcode- und QR-Scanner (z. B. *Barcoo* [20]) können die entsprechenden Codes gelesen werden, während *VisionHunt* [21] auf das Erkennen von Farben, Lichtquellen und -verhältnissen spezialisiert ist. Einen komplett anderen Ansatz verfolgt die Smartphone-App *Be My Eyes* [22], bei der sich Blinde und Sehbehinderte per Videochat mit einer sehenden Person, die bereit ist Hilfestellungen zu geben, verbinden lassen können.

Ein assistives System bestehend aus einer Kamera, einem Computer, einem Mikrophon und Ohrhörern, das dem Nutzer Farben und Muster von Kleidung mitteilt, wird in [29] beschrieben. Es werden elf verschiedene Farben sowie die Muster kariert, gestreift, nicht gemustert und unregelmäßig erkannt.

Bei der Aufnahme von Videos mit tragbaren Kameras, wie sie oftmals bei Assistenzsystemen für Blinde und Sehbehinderte vorkommen, kann es leicht zu unscharfen Frames kommen. Dies erschwert die Extraktion von Merkmalen erheblich. Daher wird in [46] ein Algorithmus vorgestellt, der unterscheidet, welche Frames eines Videostreams unscharf sind und welche nicht. Nur die Frames mit guter Qualität werden im Folgenden weiterverarbeitet. Folgen mehrere unscharfe Frames aufeinander, wird ein Deblurring-Algorithmus eingesetzt.

Im Gegensatz zu SSD, deren Ziel es ist, die visuelle Wahrnehmung auf einen anderen Sinn zu übertragen, wird in [48] ein künstliches visionäres System geschaffen, indem von einer Kamera aufgenommene Bilder in visuelle Signale umgesetzt und mit Hilfe von EEG (Elektroenzephalografie) auf den optischen Nerv übertragen werden.

IV. Analyse und Vergleich

Die in Kapitel III vorgestellten KBAS werden im Folgenden hinsichtlich Systemaufbau, Verfügbarkeit und Kosten, Bedienbarkeit sowie Funktionalität und Performanz analysiert und miteinander verglichen.

A. Systemaufbau

Die bereits erhältlichen Smartphone-Apps zur Unterstützung Blinder und Sehbehinderter beinhalten keine Transmission. Die Berechnungen wie Feature-Extraction und Evalution finden direkt auf dem Gerät statt. Teilweise erfolgt ein Zugriff auf eine lokale Datenbank oder ein Netzwerk, um Kontextinformationen abrufen zu können (z. B. *Barcoo*, *TapTapSee*). Die Ausgabe erfolgt im Allgemeinen per TTS. Eine Ausnahme ist die App *Be My Eyes*, bei der die mit dem Smartphone gemachten Aufnahmen nicht analysiert, sondern per Videochat an eine Person der Community weitergeleitet werden.

Andere Systeme, wie etwa *iFEPS* und *iMAPS*, verwenden die Smartphone-Kamera oder im Falle von *Expression* die Google Glass als Aufnahemgerät, übertragen die Daten, meist per Wifi, auf einen Server und führen dort aufwändige Berechnungen (Feature Extraction und Evaluation) aus. Ob Kontextinformationen über eine lokale Datenbank und/oder ein Netzwerk zur Berechnung bzw. Verifizierung der Gesichtserkennung herangezogen werden, geht aus der Literatur nicht hervor. Abb. 2 zeigt die Systemarchitektur von *iMAPS* adaptiert nach [6].

Bei den beiden Systemen *OrCam* und *FingerReader* zum Vorlesen von Text ist die Verwendung einer lokalen Datenbank zur Optical Character Recognition (OCR) möglich. Details über die verwendeten OCR-Algorithmen werden in der Literatur aber nicht genannt.

Die KBAS zur Wahrnehmung der Umgebung setzen im Allgemeinen nicht auf das Verwenden von Kontextinformationen über lokale Datenbanken oder Netzwerke. Die Videoaufnahmen werden per Kabel oder Wifi an eine Recheneinheit übertragen und dort weiterverarbeitet. Die Art und Weise der Ausgabe dieser Systeme ist sehr unterschiedlich (z. B. Audiosignale, taktile Displays, haptisches Feedback). Abb. 3 zeigt den Systemaufbau des in [31] beschriebenen tragbaren Kamerasystems, dessen Ausgabe über ein auf der Zunge angebrachtes elektrotaktiles Display erfolgt.

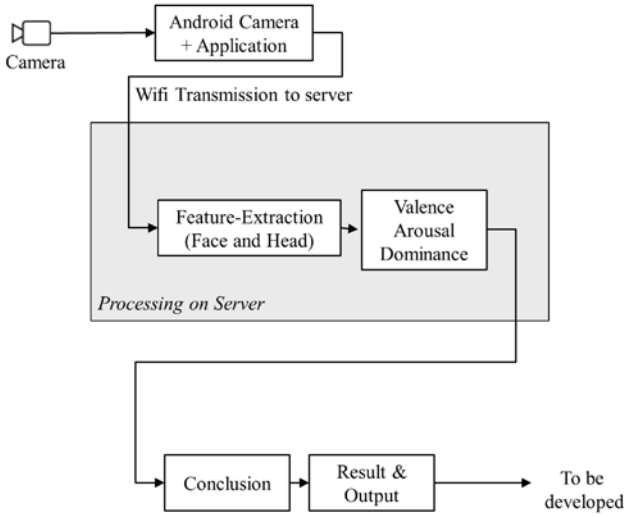


Abb. 2 System-Architektur von *iMAPS* adaptiert nach [6]

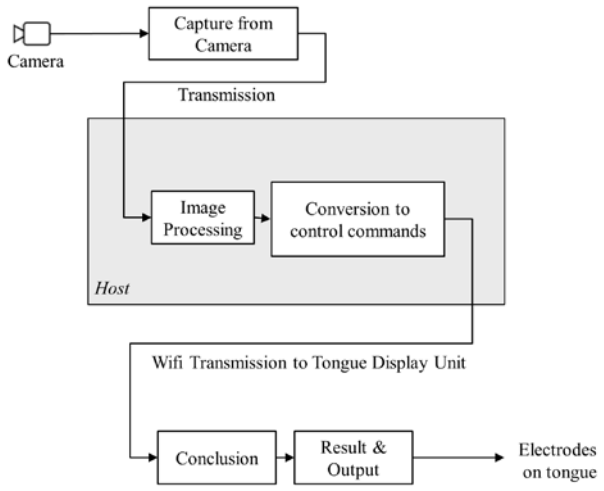


Abb. 3 System-Architektur (adaptiert) des in [31] vorgestellten Assistenzsystems

Abb. 4 zeigt die Systemarchitektur des in [29] beschriebenen KBAS, das dem Nutzer Farben und Muster von Kleidung mitteilt. Es wird weder eine lokale Datenbank noch eine Netzwerkanbindung verwendet.

Die hier vorgestellten Systeme besitzen alle eine Bilderfassungseinheit, eine komplexe Bildverarbeitungseinheit (mit Ausahme von *Be My Eyes*) mit sehr unterschiedllichen Funktionen und angemessene Ausnahmetechniken. Die Bilderfassungs- und Bildverarbeitungseinheit sind meist per Kabel oder Wifi (Transmission) miteinander verbunden. Im Rahmen der Bildverarbeitungseinheit setzen aber nur wenige Systeme, vornehmlich kommerzielle Smartphone-Apps, auf eine umfangreiche Nutzung von Kontextinformationen. Tab. I zeigt einen Vergleich verschiedener Assistenzsysteme bezüglich ihres Systemaufbaus.

B. Kosten und Verfügbarkeit

Die in Kapitel II genannten, kommerziellen Smartphone-Apps sind im Allgemeinen günstig und gut verfügbar. Einige sind lediglich für iOS, nicht aber für Android erhältlich. Oftmals, aber nicht immer, lassen sich entsprechende vergleichbare Apps für Android finden.

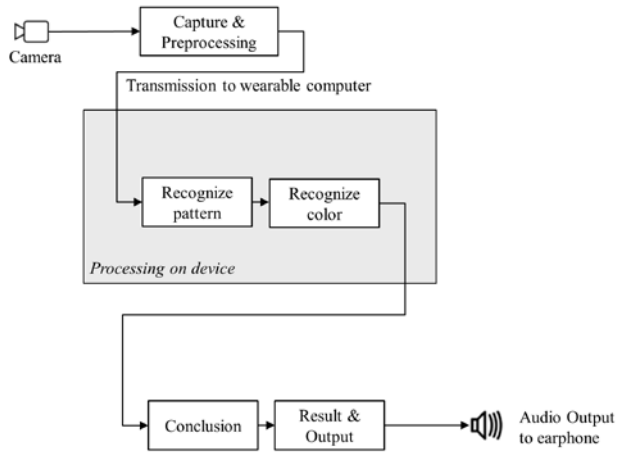


Abb. 4 System-Architektur (adaptiert) des in [29] vorgestellten Assistenzsystems

TAB. I VERGLEICH DES SYSTEMAUFBAUS VON ASSISTENZSYSTEMEN

System	Bilderfassung	Bildverarbeitung	Ausgabe
Smartphone-Apps	Smartphone-Kamera. Keine Transmission	Je nach Anwendung. Teilweise Kontext-information.	TTS
FingerReader/ OrCam	Kamera. Trans-mission per Kabel.	Optical Character Recognition (OCR). Kontextinformation möglich.	TTS
Expression/ iMAPS/ iFEPS	Google Glass bzw. Smart-phone-Kamera. Transmission per Wifi.	Erkennung von Ge-sichtszügen und Kopf-bewegungen. Kontext-information möglich.	TTS/noch nicht fest-gelegt/ Audio-signale
Systeme zur Wahrnehmung der Umgebung	Kamera. Trans-mission per Kabel oder Wifi.	Bilder verarbeiten und für Umsetzung in Ausgabeeineheit vor-bereiten. Keine Kon-textinformation.	Audio, taktile Displays, haptische Rück-meldung.
SmartSpecs	3D-Tiefen-kamera. Trans-mission per Kabel.	Filtern des Hinter-grundes, Verstärkung der Kontraste. Keine Kontextinformation.	Vision
Crosswatch	Smartphone-Kamera. Keine Transmission.	Vergleich einer Panoramaaufnahme mit Google Maps Satellitenbildern (Kontextinformation).	Nicht angegeben

Von den genannten mobilen Assistenzsystemen mit Kameraunterstützung sind bislang nur sehr wenige kommerziell verfügbar; meist werden lediglich Prototypen für die Forschung gebaut. Im Allgemeinen ist davon auszugehen, dass mobile Assistenzsysteme aufgrund ihrer speziellen und teils aufwändigen Hard- und Software sehr viel teurer sein werden als Smartphone-Apps. Die in einigen Teilen der USA erhältliche *OrCam* kostet derzeit 3500 US-Dollar, wird dafür aber von einem ausgebildeten *OrCam*-Trainer geliefert, der bei der Inbetriebnahme unterstützt. Die Sehhilfe *SmartSpecs* soll ab Frühjahr 2016 für unter 1000 US-Dollar verfübar sein; über die Orte der Verfügbarkeit liegen keine Informationen vor. Eine Ausnahme bildet das SSD *vOICe*, dessen Software über die Homepage kostenlos für Windows und Android verfügbar ist; lediglich Standardhardware muss vorhanden sein.

C. Bedienbarkeit

Die Bedienbarkeit von Assistenzsystemen für Blinde und Sehbehinderte ist ein wichtiges Thema, da aufgrund der Eingeschränktheit bzw. des Fehlens des Sehsinns alternative Bedienmöglichkeiten geschaffen werden müssen.

Smartphone-Apps basieren meist auf VoiceOver in iOS bzw. TalkBack in Android und setzen auf eine TTS-Ausgabe. In einzelnen Fällen, wie etwa *iFEPS*, werden Audiosignale als Ausgabe verwendet.

Die mobilen Assistenzsysteme verwenden unterschiedliche Ausgabemethoden. Dabei wird TTS ebenso eingesetzt wie Audio- und haptiles Feedback. TTS bietet den Vorteil, dass es

leicht verstanden werden kann; auch der Inhalt von taktilen Displays kann von Blinden und Sehbehinderten im Allgemeinen schnell erfasst werden. Um die Systeme zur Wahrnehmung der Umgebung sinnvoll nutzen zu können, ist hingegen ein längeres Training nötig, um die entsprechenden Signale mit dem Hör- bzw. Tastsinn aufnehmen und entsprechend weiterverarbeiten zu können. Der Vorteil dieser Systeme liegt in der umfassenden Information über die Umgebung, die dem Nutzer nähergebracht werden kann. Das Nutzen des Tastsinns bietet den Vorteil, dass der gerade für Blinde und Sehbehinderte wichtige Hörsinn nicht weiter als ohnehin in Anspruch genommen werden musss.

D. Funktionalität und Performanz

Die meisten der beschriebenen Smartphone-Apps und mobilen Assistenzsysteme erfüllen eine bestimmte Aufgabe bzw. mehrere kleinere Aufgaben, die einem Themengebiet zugeordnet werden können, wie etwa die App *VisionHunt*, die nicht nur Farben, sondern auch Lichtquellen und -verhältnisse erkennt. Oftmals haben die beschriebenen Systeme, die mindestens aus einer Kamera und einer Recheneinheit bestehen, das Potential vielfältige Funktionen zu übernehmen. Dies wird jedoch meist nicht ausgeschöpft, da oftmals lediglich Prototypen für einen bestimmten Forschungszweck gebaut werden.

Eine Ausnahme bildet das kommerzielle System *OrCam*, das zusätzlich zu seiner Hauptaufgabe, dem Erkennen und Vorlesen von Text, Objekte und Gesichter wiedererkennen kann; zudem sind weitere Anwendungen, wie das Erkennen von Busnummern, in der Entwicklung.

Die Systeme zur Wahrnehmung der Umgebung haben natürgemäß keine weiteren Funktionen, da ihr Ziel darin besteht, das visuelle Verständnis der Umgebung auf andere Sinne zu übertragen.

Die Smartphone-Apps sind den mobilen Assistenzsystemen im Bezug auf Performanz im Allgemeinen unterlegen. Das liegt daran, dass die Berechnungen der Apps meist auf dem Smartphone direkt stattfinden, während mobile Assistenzsysteme die Kameradaten auf einen Server oder direkt an einen leistungsfähigen Rechner übertragen und dort aufwändige Berechnungen durchführen können. So brauchen die Apps zum Vorlesen von Text meist mehrere Sekunden zur Extraktion des Textes aus einem Bild, während die *OrCam* ein Vorlesen in Echtzeit ermöglicht.

E. Vergleichsübersicht

Tab. 2 fasst den Vergleich einiger Systeme hinsichtlich der untersuchten Kriterien zusammen.

V. Assistenzsystemaufbau für Blinde und Sehbehinderte an der HFU

Ausgehend von den dargestellten KBAS für Blinde und Sehbehinderte, wird an der Hochschule Furtwangen ein System bestehend aus einer Kamera, einem Smartphone und einer Cloud-Anwendung entwickelt. Es wird massentaugliche Standardhardware eingesetzt, um die Kosten zu begrenzen. Der offene Aufbau ermöglicht es, vielfältige Anwendungsszenarien zu generieren und zu implementieren und das System stetig weiterzuentwickeln.

Abb. 5 zeigt einen Überblick des zu entwickelnden Systems und dessen Module. Auf der linken Seite ist die Hardware, eine Kamera, Ohrhörer bzw. Hörgerät und ein Smartphone, zu sehen. Lokal auf dem Smartphone finden Bildverarbeitungs- und Navigationsberechnungen sowie die Generierung von Ausgabetext statt. Das Smartphone ist mit der Cloud verbunden, die auf der rechten Seite abgebildet ist. Aufwändige Bildverarbeitungs- und Navigationsberechnungen können hierhin ausgelagert werden. Darüber hinaus werden Module zur Sicherung der Privatsphäre, zur individuellen Profilbildung und zur Interpretation externer Informationen entwickelt. Letzteres greift dabei auf das Internet zu, um relevante Kontextinformationen von verschiendenen Quellen, wie etwa Nachrichtenportalen, abgreifen zu können. Unter Beachtung der Persönlichkeitsrechte des Nutzers kann eine Benachrichtigung an Verwandte oder einen Pflegedienst gesendet werden, falls sich der Nutzer in potentieller Gefahr befindet. Für die Cloud-Dienste wird die von der HFU entwickelte AAL-Plattform [49] verwendet und erweitert.

Die Hauptinnovationen des KBAS sind:

- *Domänenspezifische Bilderkennung mit Einbindung externer Informationen zur Navigationsunterstützung:* Es werden Bildverarbeitungsalgorithmen für spezielle Erkennungen entwickelt. Um diese zu verbessern, werden externe Zusatzinformationen in der Cloud vorab verarbeitet und Navigationsmethoden zur Verfügung gestellt. Existierende Algorithmen können durch das Beachten der zusätzlich vorhandenen Informationen teils stark vereinfacht, beschleunigt und robuster gemacht werden.
- *Sozialwissenschaftliche Begleitung:* Die Akzeptanz und Bedienbarkeit des KBAS durch die Zielgruppe wird ebenso erforscht wie die Steigerung von Teilhabe und eigenständiger Mobilität. Fragen bzgl. Ethik, Recht und Sozialpolitik werden diskutiert.
- *Automatische und individuelle Profilbildung:* Die Präferenzen des Nutzers werden berücksichtigt.
- *Erweiterbare, privacy-enhanced AAL-Plattform:* Erweiterung der vorhandenen AAL-Plattform um PETs (Privacy Enhancing Technologies), um den Schutz der Privatsphäre sicherzustellen. Eine intelligente Zugriffsregelung erlaubt z. B. die Nutzerlokalisierung nur zur Stabilisierung der Bilderkennung, es sei denn, die Person wird vermisst. Durch einfache Erweiterungsmöglichkeiten der AAL-Plattform werden neue Anwendungen ermöglicht.

VI. Zusammenfassung

Es wurden vielfältige Smartphone-Apps und mobile Assistenzsysteme beschrieben, die blinde und sehbehinderte Menschen im Alltag unterstützen.

TAB. II VERGLEICH VON ASSISTENZSYSTEMEN

System	Systemaufbau	Kosten/Verfügbarkeit	Bedienbarkeit	Funktionlität/ Performanz
Smartphone-Apps	Keine Transmission. Teilweise Kontextinformation.	Günstig und verfügbar.	Leicht zu lernen, meist auf TTS beschränkt.	Meist eine Funktion. Langsam.
FingerReader	Kabel-Transmission. Evtl. Kontextinformation.	Nicht kommerziell verfügbar.	Motorenfeedback zur Ausrichtung, TTS.	Eine Funktion. Real-time.
OrCam	Kabel-Transmission. Evtl. Kontextinformation.	Für 3500 $ in Teilen der USA verfügbar.	Zeigen auf Text, TTS.	Vielfältige Funktionen. Real-time.
Expression/*iMAPS*/ *iFEPS*	Wifi-Transmission. Evtl. Kontextinformation.	Nicht kommerziell verfügbar.	Audioausgabe für *iFEPS* muss erlernt werden, ansonsten TTS.	Eine Funktion. Real-time.
Systeme zur Wahrnehmung der Umgebung	Transmission per Kabel oder Wifi. Keine Kontextinformation.	vOICe ist kostenfrei erhältlich. Die anderen sind nicht kommerziell verfügbar.	Oftmals Training nötig. Bei Audio zusätzliche Belastung des Hörsinns.	Eine Funktion, aber umfassende Wahrnehmung. Real-time.
SmartSpecs	3D. Transmission per Kabel. Keine Kontextinformation.	Ab Frühjahr 2016 für unter 1000 $ erhältlich. Orte der Verfügbarkeit nicht bekannt.	Kein Training nötig, Visionsausgabe.	Eine Funktion, aber umfassende Wahrnehmung. Real-time.

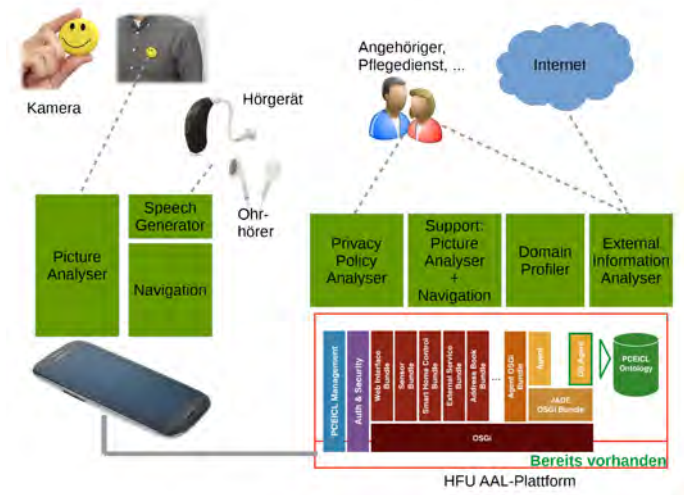


Abb. 5 System-Überblick des zu entwickelnden Assistenzsystems

Die Einsatzgebiete umfassen hierbei das Vorlesen von Text, das Erkennen von Gesichtern und Objekten, die Wahrnehmung der Umgebung sowie Navigation und Hindernis-erkennung.

Die Hauptunterschiede im Systemaufbau liegen in der Transmission zu einer externen Recheneinheit, der Nutzung von Kontextinformationen und der Art und Weise der Ausgabe. Darüber hinaus sind die jeweiligen Bildverarbeitungseinheiten auf die zu lösende Problemstellung ausgerichtet und unterscheiden sich damit fundamental. Die Kosten von Smartphone-Apps sind gering, während mobile Assistenzsysteme recht teuer und bislang selten verfügbar sind.

Auf eine angemessene Bedienbarkeit wird bei allen Systemen geachtet. Die Systeme zur Wahrnehmung der Umgebung erfordern dabei ein gewisses Training, liefern dafür aber umfassende Rückmeldungen. Um den im Alltag stark beanspruchten Hörsinn zu entlasten, wird vermehrt an den Tastsinn nutzenden Ausgabemöglichkeiten geforscht. Mit Ausnahme von *OrCam* erfüllen die vorgestellten Assistenzsysteme und Smartphone-Apps meist nur eine Aufgabe, obwohl oftmals das Potential für vielfältige Anwendungen gegeben ist. Mobile Assistenzsysteme sind den Smartphone-Apps im Bezug auf Performanz im Allgemeinen überlegen, da sie die Bilddaten auf leistungsstarke Rechnereinheiten übertragen.

Das an der HFU geplante KBAS für Blinde und Sehbehinderte, bestehend aus einer Kamera, einem Smartphone und einer Cloud-Anwendung, setzt auf einen umfassenden Einsatz von Kontextinformationen. Weitere Innovationsfelder des Systems sind eine automatische Profilbildung, die Erweiterung einer privacy-enhanced AAL-Plattform und sozialwissenschaftliche Begleitung zur Sicherstellung von Akzeptanz und Bedienbarkeit durch die Zielgruppe.

LITERATUR

[1] N. Huber. *Ohne Bilder im Bilde: eine qualitative Studie zur Mediennutzung und Medienbewertung von blinden Menschen in Deutschland*. LIT Verlag Münster, 2004.

[2] *OrCam*. http://www.orcam.com. Eingesehen am 05.10.2015.

[3] R. Shilkrot, J. Huber, C. Liu, P. Maes und S. C. Nanayakkara. „FingerReader: A wearable device to support text reading on the go". In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*. 2014, S. 2359 – 2364.

[4] A. S. M. Anam, S. Alam und M. Yeasin. „Expression: a google glass based assistive solution for social signal processing". In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 2014, S. 295 – 296.

[5] M. I. Tanveer, A. S. M. Anam, M. Yeasin und M. Khan. „Do you see what I see?: designing a sensory substitution device to access non-verbal modes of communication". In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 2013.

[6] A. K. M. M. Rahman, M. I. Tanveer, A. S. M. I. Anam und M. Yeasin. „IMAPS: A smart phone based real-time framework for prediction of affect in natural dyadic conversation". In *Visual Communications and Image Processing (VCIP)*. IEEE, Nov. 2012, S. 1 – 6.

[7] *LookTel Recognizer*. http://www.looktel.com/recognizer. Eingesehen am 05.10.2015.

[8] *LookTel Money Reader*. http://www.looktel.com/moneyreader. Eingesehen am 05.10.2015.

[9] *TapTapSee*. http://www.taptapseeapp.com/. Eingesehen am 05.10.2015

[10] *Smart Specs; VA-ST, Oxford, UK*. http://www.va-st.com/smart-specs/. Eingesehen am 05.10.2015.

[11] *OIWOB – „Orientieren, Informieren, Warnen. Orientierungshilfe für Blinde“*. http://www.oiwob.de/. Eingesehen am 05.10.2015.

[12] M. Jeon, N. Nazneen, O. Akanser, A. Ayala-Acevedo und B. Walker. „Listen2dRoom: helping blind individuals understand room layouts“. In *CHI'12 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2012, S. 1577 – 1582.

[13] D. J. Brown, T. Macpherson und J. Ward. „Seeing with sound? Exploring different characteristics of a visual-to-auditory sensory substitution device”. In *Perception* 40.9 (2011), S. 1120 – 1135.

[14] P. B. Meijer. „An experimental system for auditory image representations”. In *IEEE Transactions on Biomedical Engineering* 39.2 (1992), S. 112 – 121.

[15] *Seeing with sound.* http://www.seeingwithsound.com/. Eingesehen am 06.10.2015.

[16] *Blindsquare.* http://blindsquare.com/. Eingesehen am 06.10.2015.

[17] *LookTel Breadcrumbs*. http://www.looktel.com/breadcrumbs. Eingesehen am 06.10.2015.

[18] *Imago Project, Ambient Assisted Living (AAL), program call 4.* http://www.perfectmanage.eu/userfiles/2252/files/imago%20oct%2014%20website%20presentation.pdf. Eingesehen am 06.10.2015.

[19] V. N. Murali und J. M. Coughlan. „Smartphone-based crosswalk detection and localization for visually impaired pedestrians”. In *Multimedia and Expo Workshops (ICMEW), 2013 IEEE International Conference on.* IEEE, 2013, S. 1 – 7.

[20] *Barcode und QR Scanner barcoo*. http://www.barcoo.com/barcode-scanner-qr-code-reader-download. Eingesehen am 07.10.2015.

[21] *VisionHunt*. https://itunes.apple.com/de/app/visionhunt/id404834029?mt=8. Eingesehen am 07.10.2015.

[22] *Be My Eyes*. http://www.bemyeyes.org/. Eingesehen am 07.10.2015.

[23] D. Aguerrevere, M. Choudhury und B. Armando. „Portable 3D sound/sonar navigation system for blind individuals”. In *International latin american and Caribbean conference for engineering and technology*. 2004, S. 1 – 6.

[24] *KFNB Reader.* http://www.knfbreader.com/. Eingesehen am 08.10.2015.

[25] *Blindsight Text Detective.* http://blindsight.com/. Eingesehen am 08.10.2015.

[26] *ABBYY Textgrabber + Translator.* http://www.abbyy.com/textgrabber/. Eingesehen am 08.10.2015.

[27] J. A. Russel. „Evidence of convergent validity on the dimensions of affect”. In *Journal of Personality and Social Psychology* 36.10 (1978), S. 1152 – 1168.

[28] P. Ekman und W. Friesen. „Facial action coding system”. In *Consulting Psychologists Press*, Standford University, Palo Alto, 1977.

[29] X. Yang, S. Yuan und Y. Tian. „Assistive clothing pattern recognition for visually impaired people”. In *Human-Machine Systems, IEEE Transactions on* 44.2 (2014), S. 234 – 243.

[30] P. Bach-y-Rita und K. A. Kaczmarek. „Tongue placed tactile output device”, U.S. Patent No. 6,430,450. 6 Aug. 2002.

[31] T. H. Nguyen, T. H. Nguyen, T. L. Le, T. T. H. Tran, N. Vuillerme und T. P. Vuong. „A wearable assistive device for the blind using tongue-placed electrotactile display: Design and verification”. In *Control, Automation and Information Sciences (ICCAIS), 2013 International Conference on*. IEEE, 2013, S. 42 – 47.

[32] A. Anbuvenkatesh, S. Pownkumar, R. Abinaya, M. Rajaram und S. Sasipriya. „An efficient retinal art for blind people”. In *Computational Intelligence and Computing Research (ICCIC), 2014 IEEE International Conference on*. IEEE, 2014, S. 1 – 4.

[33] K. Moller, F. Toth, L. Wang, J. Moller, K. O. Arras, M. Bach, S. Schumann und J. Guttmann. „Enhanced perception for visually impaired people”. In *Bioinformatics and Biomedical Engineering , 2009. ICBBE 2009. 3rd International Conference on*. IEEE, 2009, S. 1 – 4.

[34] R. Velazquez, E. Fontaine und E. Pissaloux. „Coding the environment in tactile maps for real-time guidance of the visually impaired”. In *Micro-NanoMechatronics and Human Science, 2006 International Symposium on*, IEEE, 2006, S. 1 – 6.

[35] S. Horvath, J. Galeotti, B. Wu, R. Klatzky, M. Siegel und G. Stetten. „FingerSight: Fingertip haptic sensing of the visual environment”. In *Translational Engineering in Health and Medicine, IEEE Journal of* 2 (2014), S. 1 – 9.

[36] S. A. Bouhamed, J. F. Eleuch, I. K. Kallel und D. S. Masmoudi. „New electronic cane for visually impaired people for obstacle detection and recognition“. In *Vehicular Electronics and Safety (ICVES), 2012 IEEE International Conference on*. IEEE, 2012, S. 416 – 420.

[37] J. M. H. du Buf, J. Barroso et al. „The SmartVision navigation prototype for blind users”. 2011.

[38] Y. Wei und M. Lee. „A guide-dog robot system research for the visually impaired”. In *Industrial Technology (ICIT), 2014 IEEE International Conference on.* IEEE, 2014, S. 800 – 805.

[39] R. Tapu, B. Mocanu und T. Zaharia. „Real time static/dynamic obstacle detection for visually impaired persons”. In *Consumer Electronics (ICCE), 2014 IEEE International Conference on*. IEEE, 2014, S. 394 – 395.

[40] D. K. Liyanage und M. U. S. Perera. „Optical flow based obstacle avoidance for the visually impaired“. In *Business Engineering and Industrial Applications Colloquium (BEIAC), 2012 IEEE*. IEEE, 2012, S. 284 – 289.

[41] S. Pundlik, M. Tomasi und G. Luo. „Collision detection for visually impaired from a body-mounted camera“. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*. IEEE, 2013, S. 41 – 47.

[42] S. Wang und Y. Tian. „Detecting stairs and pedestrian crosswalks for the blind by RGBD camera”. In *Bioinformatics and Biomedicine Workshops (BIBMW), 2012 IEEE International Conference on*. IEEE, 2012, S. 732 – 739.

[43] Y. H. Lee und G. Medioni. „A RGB-D camera based navigation for the visually impaired”. In *RSS 2011 RGBD: Advanced Reasoning with Depth Camera Workshop*. 2011, S. 1 – 6.

[44] J. M. S. Martinez und F. E. Ruiz. „Stereo-based aerial obstacle detection for the visually impaired”. In *Workshop on Computer Vision Applications for the Visually Impaired*. 2008.

[45] H. Fernandes, P. Costa, V. Filipe, L. Hadjileontiadis und J. Barroso. „Stereo vision in blind navigation assistance“. In *World Automation Congress (WAC), 2010.* IEEE, 2010, S. 1 – 6.

[46] L. Tian, Y. Tian und C. Yi. „Detecting good quality frames in videos captured by a wearable camera for blind navigation”. In *Bioinformatics and Biomedicine (BIBM), 2013 IEEE International Conference on*. IEEE, 2013, S. 334 – 337.

[47] *Blinden- und Sehbehindertenverein Südbaden*. http://www.bsvsb.org/. Eingesehen am 10.10.2015.

[48] S. Z. Sarwar, I. Manarvi, M. S. Aslam, A. Ishaque und M. Azeem. „Noninvasive imaging system for visually impaired people“. *In Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on.* Vol. 9. IEEE, 2010, S. 1 – 6.

[49] H. Kuijs, C. Rosencrantz und C. Reich. „A context-aware, tntelligent and flexible ambient assisted living platform architecture”. In *CLOUD COMPUTING 2015 (2015): 85.*

Autonome Roboterschwärme in geschlossenen Räumen

Roberto Ciano
Hochschule Furtwangen University
Fakultät Informatik
78120 Furtwangen, Germany
roberto.ciano@hs-furtwangen.de

Dimitrij Klesev
Hochschule Furtwangen University
Fakultät Informatik
78120 Furtwangen, Germany
dimitrij.klesev@hs-furtwangen.de

***Zusammenfassung*—In diesem Beitrag wird eine Server-basierte Architektur zur Koordination mehrerer unabhängiger Roboter erläutert, die im Rahmen zweier Bachelor-Arbeiten entwickelt und prototypisch implementiert wurde. Diese Architektur dient zur Steuerung autonom agierenden Roboterschwärmen und beruht auf Funktionalitäten zur Objekterkennung und -verfolgung, welche mit Hilfe von OpenCV realisiert sind. Dies ermöglicht die Positionsbestimmung der Roboter im geschlossenen Raum über ein Kamerasystem und ein Protokoll zur Kommunikation zwischen den Robotern und einem Server. Dieser verwaltet die Positionsinformationen und steuert die Roboter. Die Roboter fragen ihre Position beim Server ab bzw. erhalten umgekehrt Kommandos, die sie autonom umsetzen. Die beiden Komponenten sind skalierbar in den Dimensionen "Anzahl der Roboter" und "Fläche, in der sich die Roboter bewegen können". Zudem ist die Erweiterung in den dreidimensionalen Raum möglich, z.B. zur Steuerung von Drohnen.**

***Index Terms*—Munkres, autonome Systeme, Objekterkennung, Homographie, Stereo Vision, XBee, Roboterschwärme**

Ciano, Klesev
Oktober 10, 2015

I. Einführung

Tracking-Technologien wie z.B. GPS ermöglichen es, Outdoor-Roboter mit begrenztem Aufwand automom zu steuern. Doch in geschlossenen Räumen erweist sich die Positionsbestimmung als deutlich problematischer, da keine allgemein verfügbare Basistechnologie zur hinreichend präzisen Ortung verfügbar ist. Deshalb wurden in zwei Abschlußarbeiten untersucht, welche Architektur sich zur Steuerung autonom beweglicher Roboter in geschlossenen Räumen eignet. Auf der einen Seite wurde ein Lösungskonzept entwickelt und umgesetzt, mit dem sich Objekte über Kameras erkennen und tracken lassen, und welches ein Mapping von Kamera- zu Weltkoordinaten erlaubt. Daneben ist das System in der Lage, die Roboter mit Hilfe definierter Kommandostrukturen zu steuern. Der andere Teil beschäftigt sich mit dem Aufbau und der Programmierung einer Roboter-Plattform, die von diesem System erfasst und angesteuert werden kann. In den beiden Arbeiten wird mit Infrarotsignalen zur Objekterkennung und mit XBee-Modulen zur Kommunikation gearbeitet.

Ein Kamerasystem erfasst dabei die Infrarotstrahlung, die von einer LED auf dem Roboter emittiert wird. Diese Informationen werden genutzt, um dem Roboter eine Position in Weltkoordinaten zuzuweisen. Dieser Ansatz bietet den Vorteil, dass eventuell vorhandene Überwachungskameras genutzt werden können, wenn das System in realen Szenarien eingesetzt wird.

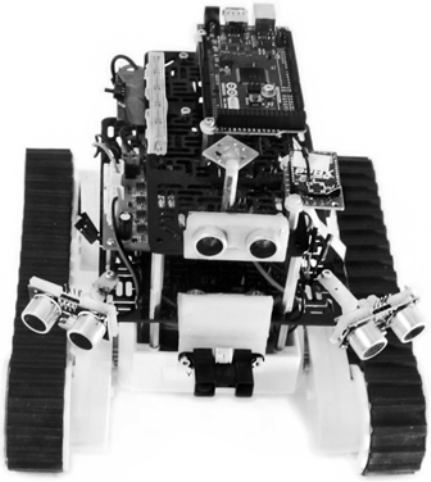


Abbildung 1. Prototyp des Roboters

II. Konzept und Architektur

Die entwickelte Architektur basiert roboterseitig auf den weit verbreiteten Arduino MEGA Boards, welche als Steuereinheit eingesetzt werden, und serverseitig auf OpenCV. Als Kamera-Server werden Raspberry Pi Systeme eingesetzt. Die Video-Ströme werden als MJPEG an den Server übertragen. Es wurde eine x86-basierte Workstation für die Server-Core Einheit verwendet. Insgesamt wurden drei baugleiche Roboter entworfen und programmiert.

A. High-Level Architektur des Roboters

Der Roboter besteht aus einem kettengetriebenen Chassis, auf dem sich über zwei Ebenen verteilt die Steuerungselektronik und die Peripherie befinden (siehe Abb. 2). Auf der oberen Ebene befinden sich der *Arduino Mega ADK* (1), die Infrarot LED (2), das *XBee Modul* (3), ein Ultraschallsensor vom Typ *HC-SR 04* (4) und ein digitaler Schalter (5). Der Arduino ist die zentrale Steuereinheit des Roboters. Er ist für die Auswertung der Sensordaten, die

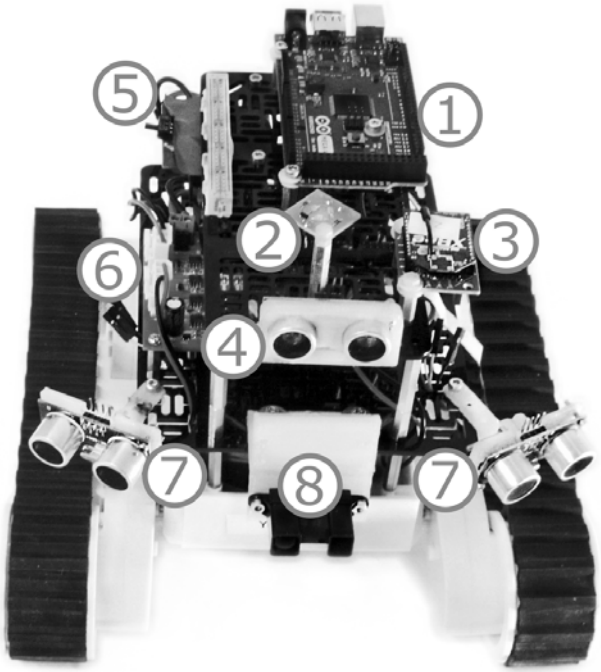


Abbildung 2. Der Aufbau des Roboters

Kommunikation mit dem Server und anderen Robotern und die Steuerung der Motoren zuständig. Die Infrarot-LED dient der Erkennung des Roboters auf den Kamerabildern. Das XBee-Modul wird vom Arduino für die Kommunikation verwendet. Der Ultraschallsensor wird zusammen mit zwei weiteren, im Folgenden beschriebenen Ultraschallsensoren für die Hinderniserkennung eingesetzt. Mit dem digitalen Schalter lässt sich der Roboter auf einfache Art und Weise ein- und ausschalten.
Auf der mittleren Ebene sind der Spannungswandler, das *Daguboard* (6), die beiden weiteren Ultraschallsensoren vom Typ *HC-SR 04* (7) und ein Infrarotsensor (8) installiert. Der Spannungswandler stellt eine stabile und geglättete Versorgungspannung der elektronischen Bauteile über einen Akku sicher. Das Daguboard übersetzt die abstrakten Kommandos zur Steuerung der Motorgeschwindigkeit und -drehrichtung in elektrische Impulse für die Motoren. Die Ultraschallsensoren werden zur Hinderniserkennung genutzt. Der Infrarotsensor kommt nicht zum Einsatz und kann für spätere Projekte genutzt werden.

Der Arduino steuert den Roboter mit einer prozedural aufgebauten Software. Die main()-Routine ist schematisch als Ablaufdiagramm in Abb. 3 zu sehen. Zuerst erfolgt stets die Prüfung auf ein Hindernis mithilfe der Ultraschallsensoren. Wird ein Hindernis erkannt, stoppt der Roboter die Motoren und begibt sich in einen Ruhezustand. Andernfalls prüft er jeweils, ob seine Zielkoordinate gesetzt wurde, und ob er seine aktuelle Position kennt. Ist eine der beiden Bedingungen falsch, sendet er in kurzen zeitlichen Abständen Anfragen diesbezüglich zum Server. Hat er alle benötigten Informationen vorliegen, berechnet der Roboter seinen aktuellen Fahrtwinkel aus seiner aktuellen und seiner letztbekannten Position, relativ zur Zielkoordinate. Überschreitet dieser Winkel einen gewissen Toleranzwert, versucht der Roboter durch die Verringerung der Geschwindigkeit auf einem Motor, den Fahrtwinkel entsprechend zu korrigieren. Abschließend wird geprüft, ob sich der Roboter bereits an seiner Zielkoordinate befindet. Ist dies der Fall, werden die Motoren gestoppt. Am Ende der main()-Routine wird stets auf neu eingegangene Nachrichten des XBee-Moduls geprüft, die dann entsprechend verarbeitet werden.
Zur softwareseitigen Auswertung der Ultraschallsensoren auf dem Arduino kommt die Bibliothek *NewPing*[1] zum Einsatz. Sie ermöglicht eine effiziente und fehlertolerante Auswertung der gemessenen Rohwerte über eine einfache API. Um die XBee-Module einfach nutzen zu können, wird die Bibliothek *xbee-arduino*[2] verwendet. Sie bietet zahlreiche Methoden für die Nutzung der XBee-typischen API-Betriebsart inklusive einer einfachen Handhabung von Sendeberichten.
Der Zustand des Roboters ist in einer C-Struktur festgehalten, der XBee wird mit Hilfe der Bibliothek als Objekt repräsentiert. Die restliche Funktionalität ist in Funktionen ausgelagert, die von der main()-Funktion aus aufgerufen werden. Da der Arduino keine echten Interrupts unterstützt, werden die Sensoren und das XBee regelmäßig abgefragt (polling).

B. Server-Architektur

Die Architektur des Servers besteht aus mehreren Komponenten. Dabei kann diese jedoch auf die wichtigsten reduziert werden, dies wird in Abb. 4 aufgezeigt.

Dabei müssen die beiden Komponenten *KalmanFilter* und *Munkres* nicht zwingend gemeinsam eingesetzt werden. In vorliegenden Fall kommt nur der Algorithmus von Munkres zur Anwendung. Es bleibt jedoch die Möglichkeit, alternativ das Modul KalmanFilter zu verwenden, sofern die Implementierung stabilisiert wird.

Die Munkres-Funktion löst das zentrale Problem der Zuordnung zwischen erkannten LED-Positionen und den hierfür relevanten Roboter-Instanzen. Die KalmanFilter-Funktion würde dabei die Zuordnung weiter optimieren. Dies wäre insbesondere bei höherer Geschwindigkeit der Roboter wichtig, wenn die FPS-Rate (frames per second) nicht erhöht werden kann. Es ist zu beachten, dass jedoch bei einer 3D-Erweiterung der Architektur das Modul KalmanFilter eingesetzt werden muss, da die Objekte sich überschneiden könnten.

Das gesammte System wird über die Controller-Komponente verwaltet. So interagieren auch andere Komponenten bzw. Objekte mit dieser. Abb. 5 veranschaulicht das Zusammenspiel der verschiedenen Objekte zur Laufzeit. Die Objekte interagieren dabei über Callback-Mechanismen.

Um die Stabilität des Systems zur Laufzeit zu garantieren[3], enthalten wichtige Objekte unabhängige Thread-Objekte.

[1] https://code.google.com/p/arduino-new-ping/
[2] https://code.google.com/p/xbee-arduino/
[3] Damit ist zunächst die Ressourcennutzung gemeint

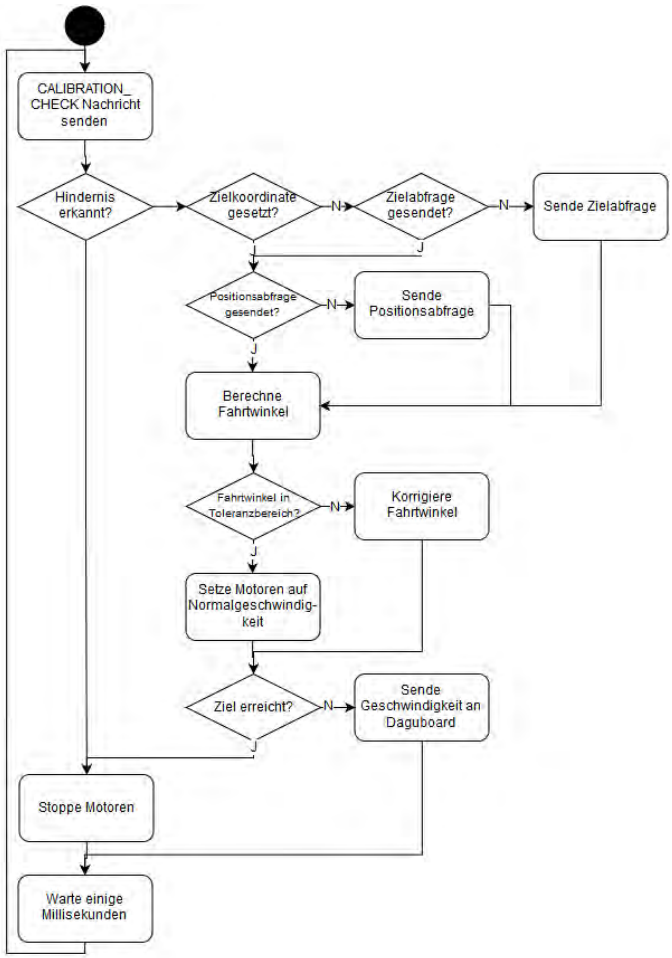


Abbildung 3. Ablauf der Hauptroutine auf dem Roboter

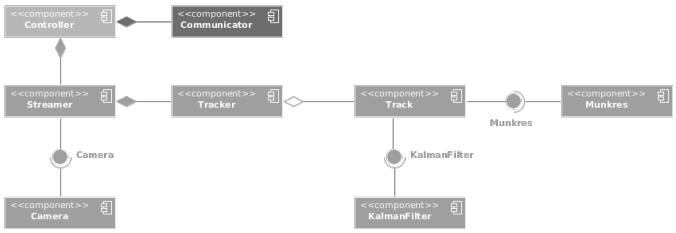


Abbildung 4. Wichtigste Komponenten der Softwarearchitektur

Listing 1 zeigt das Template für die Verwendung von Multi-Threading. Im Konstruktor wird das jeweilige Thread-Objekt initialisiert und die *__call__()*-Methode wird als Main-Loop des Threads aufgerufen.

C. Autonome Systeme

Die Roboter agieren autonom unter Berücksichtigung ihrer Umgebung und der erhaltenen Befehle. Dies bedeutet, dass die Roboter-Steuerung jeweils lokal aufgrund externer High-level Kommandos erfolgt. Hierzu wird ein zustandsbasiertes Modell auf den Robotern umgesetzt, das in Abb. 6 skizziert ist. Im implementierten Use-Case erhalten die Roboter den Befehl,

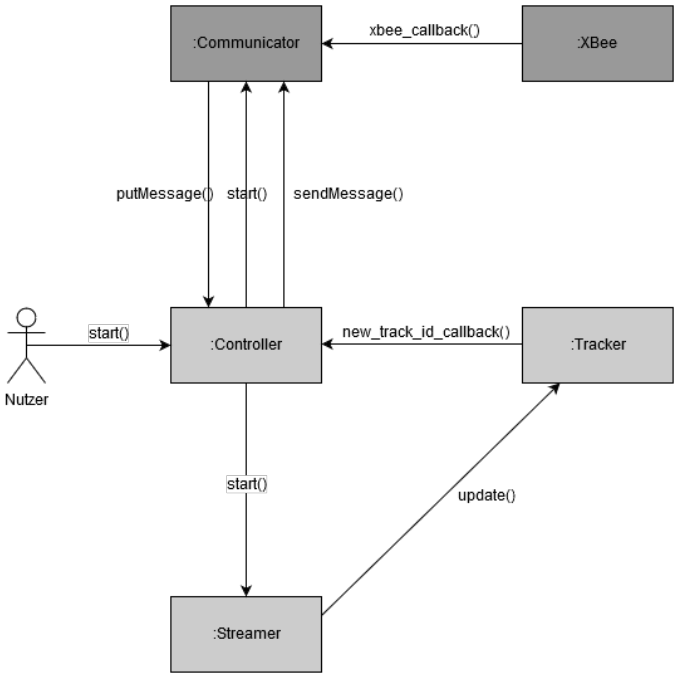


Abbildung 5. Zusammenspiel der einzelnen Klassenobjekte zur Laufzeit

```
import threading
...
class MyClass:
   def __init__(self, params):
      self.thread = threading.Thread(None, self, "
          MyClass")
      ...
   def __call__(self, *args, **kwargs):
      ...WORK...
```

Listing 1. Thread-Template

ein durch Weltkoordinaten definiertes Ziel anzufahren. Dabei sollen sie selbstständig den Weg finden und Kollisionen mit anderen Robotern vermeiden. Mithilfe ihrer Sensoren sind sie in der Lage, in der Nähe befindliche Objekte zu erkennen und entsprechend darauf zu reagieren. Im derzeitigen Stand der Implementierung wird die Fahrt jedoch unterbrochen, um eine Kollision zu vermeiden. Die Fahrtunterbrechung und die Fahrt selbst werden jeweils durch einen Zustand beschrieben. Diese Zustände finden sich im Quellcode in Form von boolschen Variablen in der Roboter-Datenstruktur.

Die angestrebte Master-Slave Architektur (siehe Abschnitt III) bedarf einer initialen Bestimmung der Master-Rolle. Der vorgeschlagene Algorithmus führt die Masterbestimmung serverseitig während der Kalbibrierungsphase durch. Die Roboter melden sich alle als Master-Kandidaten beim Server an. Der Roboter, dessen Nachricht als erste vom Server verarbeitet wird, bekommt den Zuschlag. Anschließend teilt der Server den Robotern mittels Broadcast die ID des Masters mit. Dadurch kennen die Slaves implizit die Adresse des Masters, die identisch mit dessen ID ist, und sie sind somit in der Lage, eine direkte Kommunikation mit ihm zu starten.

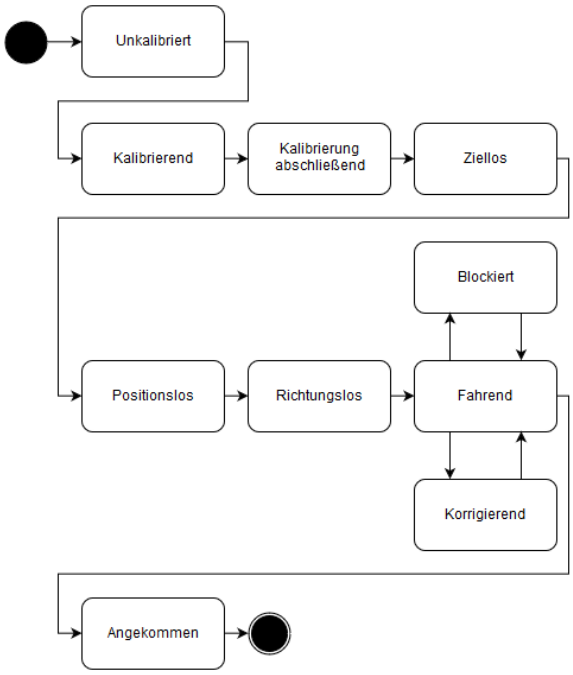


Abbildung 6. Zustandsbasierte Roboter-Steuerung

D. Kalibrierungsprozess

Bevor die Roboter durch den Server erkannt und getrackt werden können, muss eine Kalibrierung durchgeführt werden. Dadurch sind sie dem Server bekannt und können getrackt werden, was in der Folge die Positionsbestimmung des Roboters erlaubt. In diesem Prozess melden sich die Roboter beim Server an und schalten mit Hilfe eines dedizierten Kommandos jeweils ihre LED ein. Durch das sequenzielle Anschalten der LEDs kann der Server über die Kamerabilder jede LED eindeutig einem Roboter zuordnen. Der Ablauf ist in Abb. 7 dargestellt.

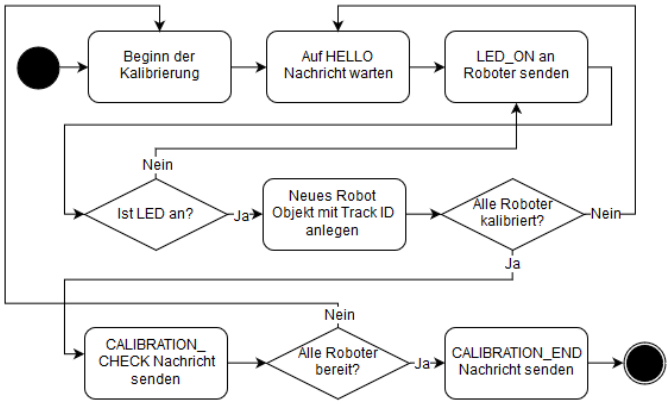


Abbildung 7. Ablauf der Kalibrierung

Zunächst werden alle Roboter mit einer Broadcast Nachricht in den Kalibrierungszustand versetzt. Daraufhin antworten alle Roboter mit einer HELLO-Nachricht. Der Server arbeitet alle HELLO-Nachrichten sequentiell ab und fordert dabei jeden Roboter auf, seine LED einzuschalten. Wurde die LED erkannt, wird mit der nächsten HELLO-Nachricht fortgefahren. Um den Prozess in dieser Form zu realisieren, benötigt der Server Kenntnis von der Anzahl der Roboter. So kann er prüfen, ob bereits alle Roboter kalibriert wurden. Ein einfaches Abarbeiten der Message Queue würde nicht immer funktionieren, da Nachrichten verzögert ankommen und den Server zu der Annahme bringen könnten, es seien bereits alle Roboter kalibriert.

E. Information und Kommunikation

Die Kommunikation zwischen den Robotern und dem Server geschieht mit XBee-Modulen der Serie 1, die in einem Star-Netzwerk organisiert sind. Der Server stellt das Gateway des Netzwerks dar. Das XBee-Protokoll zeichnet sich gegenüber WLAN durch einen sehr geringen Energieverbrauch bei einer guten Reichweite aus. Bluetooth scheidet als Protokoll aus, da es für die Kopplung kleiner Gruppen von Geräten entworfen wurde und sich nur bedingt zur Bildung eines hierarchischen Netzwerks eignet, das z.B. für das Master-Slave Szenario benötigt würde. Ein weiterer Vorteil der XBee-Module ist der sogenannte API-Modus. XBee-Module in diesem Modus können auf ein grundlegendes Protokoll für den Versand von Nachrichten zurückgreifen, der neben Sendeberichten auch weitere Merkmale wie eine signalbasierte Abstandsmessung zwischen einzelnen Modulen beinhaltet. [8] Es wurde ein Protokoll zur Kommunikation zwischen den Robotern und dem Server entworfen und implementiert. Es ist textbasiert und daher einfach einzusetzen und zu erweitern. Das Protokoll definiert ausschließlich die Struktur einer Anfrage und einer Antwort und kann deshalb in verschiedenen Nachrichtenprotokollen zum Einsatz kommen. Eine semiformale Definition kann **??** entnommen werden. Eine Anfrage sowie auch eine Antwort beginnen jeweils mit einem kurzen Schlüsselwort, welches kennzeichnet, um welchen Befehl oder um welche Information es sich handelt. Das Schlüsselwort darf nur aus Großbuchstaben aus dem ASCII-Standard bestehen. Bei einer Antwort mit Parametern werden an dieses Schlüsselwort beliebig viele ganzzahlige Parameter angehängt, die jeweils mit einem vorangestellten Doppelpunkt getrennt werden. Die Nachricht wird durch einen Newline-Charakter abgeschlossen.

```
MESSAGE = COMMAND , [ PARAMS ] , "\n"
PARAMS = PARAMS , PARAM | PARAM
PARAM = ":" , { DIGIT } , "!"
COMMAND = { LETTER }
DIGIT = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7
    " | "8" | "9"
LETTER = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "
    H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "
    P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "
    X" | "Y" | "Z"
```

Listing 2. Aufbau des Protokolls

F. Objektverfolgung

Eine der großen Herausforderungen bei der Steuerung automomer Indoor-Roboter ist die richtige Objekterkennung und -differenzierung. In der Industrie werden verstärkt Bildverarbeitungssyteme für Objekterkennung eingesetzt. [2] Dabei werden die Objekte mittels Zentralprojektion auf die Kamerasichtebene abgebildet. Hierbei muss jedoch unterschieden werden, ob eine rückwirkende Transformation in einen dreidimensionalen oder einen zwei-dimensionalen Raum erfolgt. Eine 3D-Rekonstruktion eines Objektes ist mit *Stereo Vision* möglich. Eine Ebene im R^3 kann mittels Transformation in eine andere Ebene im R^3 abgebildet werden.

$$T : R^3 \rightarrow R^3 \tag{1}$$

Wir beschränken uns jedoch auf den zweidimensionalen

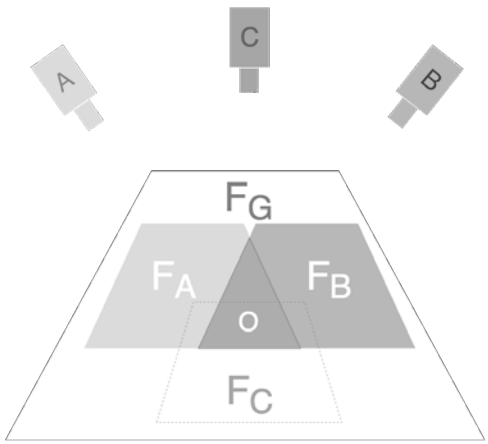


Abbildung 8. Aufbau des Multi-Kamera-Systems

Raum und wenden somit Homographie an (veranschaulicht in Abb. 8[4]). [4] Die Abstraktion in den 3D-Raum ist aufbauend auf dieser Umsetzung möglich.
Bei der Betrachtung in homogenen Koordinaten gilt für die Homographie folgendes:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{2}$$

Aufbauend auf diesem Prinzip verwenden wir ein Multi-Kamera-System, mit dem auch Stereo-Vision anwendbar ist. [6] Ein Objekt in einem Frame kann anhand seiner Konturen extrahiert werden. [3] Sich bewegende Objekte in einer Videosequenz sind somit dieselben Objekte an verschiedenen Positionen in aufeinanderfolgenden Frames. Die Zuordnung der Koordinaten und somit die Bewegungsrekonstruktion des Objektes ist mit dem Ungarischem Algorithmus[5] möglich. [7] Dieser Algorithmus aus dem Bereich der Zuordnungsprobleme ermöglicht die Bildung möglichst vieler optimaler Paare aus zwei gegebenen Mengen. Abb. 9 zeigt vereinfacht das

[4]Dabei ist die Fläche O der Kalibrierungsbereich. F_x sind die jeweiligen Kamerasichtbereiche

[5]auch Munkres genannt

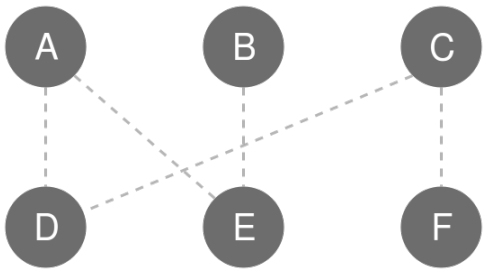


Abbildung 9. Funktionsweise des Ungarischen Algorithmus

Problem der Zuordnung auf. Einige Einheiten haben mehrere Möglichkeiten ein Paar zu bilden. Hierbei seien zwei Mengen $M_1 = \{A, B, C\}$ und $M_2 = \{D, E, F\}$ gegeben. Der Algorithmus löst dabei das Problem der optimalen Paarbildung.

$$(D, A), (E, B), (F, C) \tag{3}$$

Im vorliegenden Anwendungsfall sind es die Roboter-Instanzen und die erkannten Positionen im Bild, die einander zuzuordnen sind.

G. Hinderniserkennung und Reaktion

Damit die Roboter beim Anfahren der Zielkoordinate nicht mit anderen Robotern kollidieren, wurde eine einfache Hinderniserkennung implementiert. Mithilfe dreier Ultraschallsensoren vom Typ *HC-SR 04*, die frontal sowie zu beiden Seiten geneigt am Roboter angebracht sind, wird die Entfernung des Roboters zu ihn umgebenden Objekten gemessen. Wird ein bestimmter Mindestabstand bei mindestens einem der Sensoren unterschritten, geht der Roboter in den Zustand *Blockiert* über (siehe Abb. 6). Er bleibt so lange stehen, bis alle Sensoren wieder Werte liefern, die den Mindestabstand einhalten.

Im Rahmen der Arbeit wurde eine alternative Hindernisbehandlung ausgearbeitet, die sich insbesondere zum Umfahren von Hindernissen mit rechteckigem Grundriss eignet. Hierbei prüft der Roboter auf eine freie linke oder rechte Seite und dreht sich dann in die entsprechende Richtung um 90 Grad. Er fährt dann entlang der Kante des Hindernisses, bis es sich nicht mehr neben ihm befindet. Anschließend dreht er sich in die ursprüngliche Fahrtrichtung zurück und fährt wieder parallel zur Kante des Hindernisses, bis sich dieses hinter ihm befindet. Während dieses Vorgangs befindet er sich in einem speziellen Umfahrungszustand. In diesem Zustand vernachlässigt der Roboter die Zielanfahrt. Er setzt sie erst fort, sobald er das Hindernis umfahren hat.

III. Inbetriebnahme, Anwendungsbeispiel und Teststrategie

Das Ziel der Arbeit ist, drei Roboter zu einer dynamisch zu bestimmenden Zielkoordinate fahren zu lassen, ohne dass sie auf dem Weg miteinander kollidieren. Die Roboter sollen dabei ihre Positionen und die Zielkoordinate vom Server erhalten. Die Roboter sollen für die Ausführung des Use Cases in

einer Master-Slave Konfiguration fahren. Der Master soll dabei die Hauptkommunikation mit dem Server übernehmen und Befehle an die Slaves weiterreichen. Die Slaves sollen lediglich ihre Position vom Server direkt erfahren und andere Anfragen an den Master senden. Somit soll eine Hierarchie erreicht werden, die es den Robotern erlaubt, Aufgaben möglichst autonom und fehlertolerant zu erledigen. Die Fläche, auf der die Roboter fuhren, ist zunächst frei von Hindernissen und mindestens von einer der zwei eingesetzten Kameras ohne Einschränkungen einsehbar.
Die Kameras werden vor dem ersten Betrieb mit einem Schachbrettmuster kalibriert und danach nicht bewegt, geneigt oder rotiert. Sie haben einen Überschneidungsbereich des Sichtfeldes von ca. 2m. Insgesamt beträgt der Bereich, der von den Robotern befahren werden konnte ca. $10m^2$.
Für die Kommunikation über die XBee-Module werden die Module mit der jeweils gleichen *PAN*-ID konfiguriert. Das Modul des Servers wurde mit der *Coordinator*-Firmware bespielt, die Module der Roboter jeweils mit der *Router*-Firmware. Den *Routern* wurde jeweils eine eigene *MY*-ID zugewiesen, über die sie eindeutig im Netzwerk identifizierbar sind.

IV. Zusammenfassung und Fazit

Das in diesem Beitrag vorgestellte Konzept bietet eine Server-seitige Schnittstelle für mehrere autonome Roboter-Einheiten in einem geschlossenen Raum. Dabei wird mittels Homographie und Munkres-Algorithmus die Position des jeweiligen Roboters ermittelt. Eine Abstraktion in den 3D-Raum besteht für zukünftige Erweiterungen. Ebenfalls ist die vorgeschlagene Architektur skalierbar, sowie multi-threaded, was eine optimale Ressourcen-Nutzung ermöglicht. Das Koordinatensystem lässt sich durch das Hinzufügen weiterer Kameras erweitern, wobei der Überschneidungsbereich der Kamerabilder zur Kalibrierung dient. Die Roboter-Einheiten lassen sich durch weitere Sensorik erweitern und könnten in Zukunft mit dem echtzeitfähigem Betriebssystem ROS betrieben werden. Die Koordination der Instanzen mit Hilfe des Master-/Slave-Ansatzes ist bereits konzeptionell ausgearbeitet, muss aber noch abschließend implementiert werden.

Literatur

[1] A. Smorodov, *Hungarian algorithm + Kalman filter multitarget tracker implementation.*, https://github.com/Smorodov/Multitarget-tracker

[2] Instruments, National, *3D-Bildgebung mit NI LabVIEW*, http://www.ni.com/white-paper/14103/de/

[3] S. Suzuki, *Topological Structural Analysis of Digitized BinaryImages by Border Following* Graduate School of Electronic Science andTechnology, Shizuoka University.

[4] D. Schlesinger, *Bildverarbeitung: 3D-Geometrie*,Technische Universität Dresden, http://www1.inf.tu-dresden.de/~{}ds24/lehre/bvme_ss_2012/bv_3d.pdf

[5] R. Ciano, *Autonome Positionierung und Steuerung von Fahrrobotern in einem relativen Koordinatensystem.* Hochschule Furtwangen, 2015.

[6] D. Klesev, *Erstellung einer Server-CoreInfrastruktur und API für die Steuerungim Bereich Robotik.* Hochschule Furtwangen, 2015.

[7] M. Jünger *Aussois Workshop in Combinatori-al Optimization:50 years of integer programming 1958 -2008: from the early years to the state-of-the-art* Optimization Workshop AUSSOIS, 2008

[8] http://www.kanda.com/zigbee-wireless.html

Softwaretechnik

Beitrag der Java Modeling Language zur Qualität

Die Einordnung von JML in Bezug auf andere BISL

Nicole Pfeffermann
Fakultät Informatik
Hochschule Furtwangen
Furtwangen, Germany
nicole.pfeffermann@hs-furtwangen.de

Christine Rothfuß
Fakultät Informatik
Hochschule Furtwangen
Furtwangen, Germany
christine.rothfuss@hs-furtwangen.de

***Abstract*— Die Java Modeling Language (JML) ist eine BISL (Behavioral Interface Specification Language), welche das beabsichtigte Verhalten von Java Programmmodulen spezifiziert. Sie ist eine der meistgenutzten Spezifikationssprachen und basiert auf dem Design by Contract-Ansatz von Eiffel, dem Spezifikationsansatz von Larch und Elementen des refinement calculus [1]. Die Java Modeling Language ist mit dem Grundwissen von Java einfach zu erlernen. Weiterhin wird die Spezifikationssprache in vielen Verifikationswerkzeugen eingesetzt. Über 20 wissenschaftliche Gruppen entwickeln derzeit JML und dessen Tools und stellen ihre Arbeit in bisher 135 Abhandlungen zur Verfügung [2]. Ziel dieser Arbeit ist es, die Unterstützung von JML zur Qualitätssicherung zu validieren und die Sprache gegenüber anderen Spezifikationssprachen abzugrenzen und einzuordnen.**

In den folgenden Abschnitten wird die Java Modeling Language grundlegend beschrieben, um anschließend auf die Aspekte der Qualitätssicherung eingehen zu können. Danach wird die Definition der Behavioral Interface Specification Language beschrieben und einzelne Spezifikationssprachen genauer erläutert, um diese der Java Modeling Language gegenüberzustellen. Damit soll eine Einordnung von JML gegenüber ähnlichen Konstrukten erreicht werden. Dazu werden Gemeinsamkeiten und Gegensätze diskutiert, um den Nutzen für Unternehmen zu erörtern.

Keywords*— *Java Modeling Language, Qualitätssicherung, Behavioral Interface Specification Language, Design by Contract

I. Einleitung

Es gibt zahllose Softwareprodukte, auf welche wir uns Tag für Tag verlassen. Wenn diese Softwareprodukte fehlerhaft oder sogar lebensbedrohlich werden, dann ist das ein Hinweis auf mangelnde Qualitätssicherung bei der Erstellung von Software.

Spezifikationssprachen helfen das Designkonzept von Programmiersprachen zu verfeinern. Dabei unterstützen die Spezifikationen die Überprüfung der Anforderungen an das jeweilige Programm. Hier ist zu beachten, dass eine hohe Qualität bei der Softwareentwicklung eine große Herausforderung darstellt. Die Qualität kann nicht alleine durch Spezifikationen erreicht werden, jedoch können diese einen beachtlichen Teil dazu beitragen, welcher durch die frühzeitige Vermeidung von Fehlern geleistet wird. Dazu soll evaluiert werden, welchen Beitrag die JML leisten kann. Im Laufe der Arbeit werden andere Spezifikationssprachen und deren Definition betrachtet, um JML abzugrenzen und zu bewerten.

In den folgenden Abschnitten wird die JML und der wesentliche Design by Contract-Ansatz erläutert. Dabei wird auf eine allgemeine Beschreibung der Qualitätssicherung eingegangen und weitere Spezifikationssprachen beschrieben. Anschließend folgen ein Vergleich und die Kategorisierung der ausgewählten BISL in drei Gruppen. Das Hauptaugenmerkt liegt durchgehend auf JML, weshalb verstärkt auf deren Vor- und Nachteile eingegangen wird. Diese werden dann evaluiert und auf den Einsatz in Unternehmen übertragen.

II. JML und die Qualitätssicherung

Die Qualität von Software wird oftmals als die Abwesenheit von Fehlern beschrieben, wobei Fehler Abweichungen oder das Nicht-Erfüllen von Anforderungen sind [5].

"*Im allgemeinen wird unter „Qualität“ das Ausmaß verstanden, in dem ein Gegenstand geeignet ist, Anforderungen zu erfüllen.”* [5]

JML und dessen formale Anmerkungen unterstützen das Einhalten der funktionalen Anforderungen und helfen Fehler frühzeitig zu erkennen und zu vermeiden. Dies hilft dabei, die allgemeine Qualität des Codes zu erhöhen und das beabsichtigte funktionale Verhalten vom Programmcode sicherzustellen.

Es können formale und informale Spezifikationen im Java Programmcode als Kommentare notiert werden. Es ist jedoch zu beachten, dass nur formale Spezifikationen während der Laufzeit ausgeführt und geprüft werden [7]. Dadurch können Schwachstellen und Fehler im Programmcode aufgezeigt werden [7]. Informale Spezifikationen werden nicht zur Laufzeit ausgewertet oder automatisch überprüft [7]. Die informale Spezifikation dient lediglich der Erweiterung des Codes durch Kommentare, um eine Dokumentation zu erstellen [7]. Diese Dokumentation kann in Entwicklerteams hilfreich für die interne Kommunikation sein, denn dort wird definiert, welche Ein- und Ausgaben eine Methode hat [7]. Weiterhin bieten informale Spezifikationen eine gute Möglichkeit, formale Spezifikationen natürlichsprachlich zu notieren, wenn nicht genügend Zeit für die Implementierung der formalen Spezifikationen bleibt [7]. Daraus ergibt sich ein

Nachteil, da bei hohem Zeitdruck oft unpräzise gearbeitet wird und dadurch die informalen Spezifikationen unpräzise notiert werden. Die informale Spezifikation ist gegenüber der formalen Spezifikation ohnehin unpräzise. Nachstehend wird das Beispiel (1) für eine informale Spezifikation und Beispiel (2) für eine formale Spezifikation gezeigt:

```
//@ requires (* money is positive *);        (1)
/* @ requires money >= 0.0;                  (2)
```

Allgemein kann die Qualitätssicherung von Software in drei Bereiche eingeteilt werden: analytische, konstruktive und organisatorische Maßnahmen [8]. Die analytischen Maßnahmen befassen sich mit der aktiven Fehlersuche, wobei sich die konstruktiven Maßnahmen mit der Prävention von Fehlern, die während der Entwicklung entstehen, beschäftigen [8]. Die organisatorischen Maßnahmen bilden einen Rahmen um die analytischen und konstruktiven Maßnahmen, da sie die nötigen Voraussetzungen schaffen, um den reibungslosen Ablauf dieser zu gewährleisten [8]. Die JML ist dabei in die konstruktiven Maßnahmen einzugliedern, da die Hauptaufgabe von JML die Verhinderung von Fehlern während der Entwicklung ist. Durch die Spezifikationssprache wird es möglich, Java Code während der Entwicklung direkt zu beeinflussen und mögliche Fehler abzuwenden.

Die JML ermöglicht es, das Verhalten von Programmmodulen zu bestimmen um Mehrdeutigkeiten zu vermeiden. Weiterhin lassen sich damit Moduldesignabsichten beschreiben, um so das Design von Java-Klassen und Schnittstellen zu detaillieren [1]. Die Entwicklung der JML wurde von Gary T. Leavens und weiteren Personen der Iowa State University initiiert [9]. Ziel war es, die Java-Notation an einigen Stellen um eine aussagekräftige Spezifikation zu erweitern [2]. JML gehört zu den am meisten verbreiteten Spezifikationssprachen und baut auf die Programmiersprache Java auf. Durch die Nutzung der Java-Syntax und -Semantik ist es für Java-Programmierer einfach JML zu erlernen.

Eine Vielzahl von Verifikationswerkzeugen verwenden JML, indem sie mit dessen Hilfe Informationen auslesen, diverse Tests ausführen, Dokumentationen anzeigen, statistische Prüfungen und Laufzeitüberprüfungen vornehmen [9]. Dadurch wird die syntaktische Korrektheit spezifiziert und das Verhalten von Programmcode verifiziert. Dies geschieht durch Hinzufügen von Vor- und Nachbedingungen, Assertions und Invarianten zum Programmcode [7].

Die JML vereint den Design by Contract-Ansatz von Eiffel mit dem modellbasierten Spezifikationsansatz von Larch und Elementen von refinement calculus [1]. Die Spezifikationssprache nutzt die mathematischen Werte von Larch, um das funktionale Verhalten zu beschreiben [10]. Diese mathematischen Werte werden von JML in Model Fields beschrieben, welche nur in Assertions genutzt werden [10]. Assertions, Invarianten sowie Vor- und Nachbedingungen werden vom Design by Contract-Ansatz implementiert [10,7]. Der Design by Contract-Ansatz spaltet die Programmteile in zwei Parteien auf: den Anbieter, der Klassen bereitstellt und den Client, der die Klassen aufruft. Diese zwei Parteien schließen Verträge miteinander ab, welche aus Bedingungen für Klassen, Methoden und Attributen bestehen können. Beide Parteien gehen auf Grund der Verträge Verpflichtungen ein, welche sie erfüllen müssen. Wenn man diesen Ansatz auf JML bezieht, werden die Verpflichtungen in Vor- und Nachbedingungen, Invarianten und Assertions verfasst [7]. Während Assertions an beliebigen Stellen des Codes platziert werden können, werden Vor- und Nachbedingungen immer vor bzw. hinter einer Methode, auf die sie Bezug nehmen, geschrieben [7]. Assertions sind dazu da, den Programmzustand stetig zu überprüfen.

Die Vorbedingungen prüfen die Gegebenheiten vor dem Eintritt in eine Methode direkt nach der Parameterübergabe [7]. Somit wird eine zusätzliche Überprüfung der Voraussetzungen vor dem Aufruf überflüssig. Nachbedingungen gelten dagegen direkt nach dem Ausführen einer Methode und bieten dem Nutzer der Methode eine Garantie, dass er ein Ergebnis erhält [7]. Das vierte und letzte Element sind die Invarianten, die sich auf Objekteigenschaften beziehen [7]. Sie werden jeweils zu den Vor- und Nachbedingungen mit Hilfe einer UND-Verknüpfung hinzugefügt. Sie gelten jedoch nicht während der Ausführung einer Methode, sondern immer nur davor und danach [7]. Kommt es während der Programmausführung zu Verletzungen der Spezifikationen, wird das Programm angehalten und eine Fehlermeldung mit der verletzten Spezifikation ausgegeben.

```
public class bankAccount{

  private/* @ non_null */String bankInstitute;
  private/* @ non_null */long bankCode;
  private/* @ non_null */double bankBalance;

  /*@ public invariant !bankInstitute.equals("")
   *@ && bankCode > 0; @*/
  // @ also
  // @ensures \result == bankBalance;
  public/* @ pure @ */double getBankBalance()
  {return bankBalance;}

  /* @ also
   * @ requires money >= 0.0;
   * @ ensures bankBalance == \old(bankBalance + money);*/
  public void addMoney (double money){}
}
```

In diesem Codebeispiel sieht man Javacode, welcher durch funktionale Spezifikationen der JML erweitert wurde. Durch die Nutzung von JML wird der beschriebene Code um einige Zeilen erweitert, welche nicht aktiv die Funktionalität des Codes erweitern, sondern nur Überprüfungen des Codes darstellen. Daraus kann der vorhandene Programmcode unübersichtlicher werden. Um dieses Problem zu lösen, kann der aspektorientierte Ansatz mit dem Werkzeug AspectJML genutzt werden. AspectJML unterstützt den aspektorientierten Ansatz mit Design by Contract-Spezifikationen durch JML. Damit gelingt es die Übersichtlichkeit des Programmcodes trotz der Spezifikationen beizubehalten. Im nächsten Abschnitt

wird nun das Codebeispiel genauer erläutert und die vorherig beschriebenen Elemente erklärt.

Eine standardmäßige Vorgabe von der JML ist, dass Attribute nicht Null werden dürfen, was das *non_null* vor der Attributdeklaration beschreibt. Mit Hilfe der Invarianten wurde in diesem Beispiel ausgedrückt, dass der Name des Bankinstituts kein Leerstring sein darf und die Bankleitzahl mindestens aus einer Ziffer bestehen muss. Das Schlüsselwort *\ensure* signalisiert den Beginn einer Nachbedingung. Bei der Getter-Methode *getBankBalance* muss der Kontostand unverändert ausgegeben werden. Anhand *pure* wird ausgedrückt, dass die Methode keine Seiteneffekte hat und damit auch keine Zustände verändert. Die letzte Methode stellt die Einzahlung auf das Konto dar. Hier werden keine negativen Beträge im Übergabeparameter *money* erlaubt, da es sich um eine Einzahlung und nicht um eine Auszahlung handelt. Anhand des Schlüsselwortes *\requires* wird dies direkt nach der Parameterübergabe überprüft. Eine Nachbedingung des Codes beschreibt den Rückgabewert der Methode: den aktuellen Kontostand, welcher sich aus dem Vorzustand des Attributs *bankBalance* durch *\old* und dem eingezahlten Geld zusammensetzt.

Über das Beispiel hinaus können die, dem Design by Contract-Prinzip entsprechenden Methodenverträge (Vor- und Nachbedingungen) durch die Schlüsselwörter *\signals* und *\diverges* erweitert werden [11]. Nachbedingungen für die Ausführung im Ausnahmefall werden dabei mit *\signals* und Vorbedingungen für Nichtterminierung mit *\diverges* gekennzeichnet [11].

III. DIE BISL

Die Behavioral Interface Specification Languages bieten auf Code-Ebene formale Anmerkungen, wie Vor- und Nachbedingungen, Invarianten und Assertions [12]. Die Anmerkungen ermöglichen, das beabsichtigte Verhalten eines Programms oder einzelner Programmmodule zu beschreiben. Da die Programmmodule je nach Anwendung sehr komplex sein können helfen die ausdrucksstarken Notationen der einzelnen BISLs das Programmverhalten verständlich und prüfbar zu machen. Anhand der BISLs kann eine gute und präzise Dokumentation des Programmverhaltens erstellt und eine Steuerung der Implementierung geschaffen werden [12]. Die Implementierung kann gezielt durch die einzelnen Spezifikationen gesteuert werden, da sie durch die Anforderungen bestimmen, was implementiert werden soll und wie die Schnittstelle dazu auszusehen hat. Während dem Entwicklungsprozess dient die Spezifikation einer guten Kommunikation und klaren Vereinbarungen zwischen den verschiedenen Entwicklerteams [12]. Ziel einer BISL ist es hohe Ausdruckskraft, Vielseitigkeit und Benutzerfreundlichkeit zu vereinen [12].

Eine Behavioral Interface Specification Language ist ein Modul, das aus zwei Teilen besteht: einer Schnittstelle und einer Beschreibung des Verhaltens. Die Schnittstelle wird durch sprachenspezifische Elemente, welche durch das Modul exportiert werden, erfasst [13]. Diese Elemente bestehen aus Feld- und Methodensignaturen. Die Verhaltensbeschreibung bezieht sich auf Elemente der Schnittstelle [13].

Bei der Nutzung von Spezifikationssprachen in Verbindung mit einer automatischen Analyse und diversen Verifikationswerkzeugen können leichter Schwachstellen im Programm erkannt und anwendungsspezifische Eigenschaften erstellt werden [12]. Mit Hilfe von BISL können automatisch Testfälle generiert werden und die dargestellten Behauptungen während den Tests überprüft werden [12]. Zusätzlich können formale Spezifikationen zu Testorakeln kompiliert werden, die angeben ob ein Test bestanden oder nichtbestanden wurde [12]. Die Testorakel tragen maßgeblich dazu bei die Ausführung eines Systems zur Laufzeit zu überprüfen [12]. Formale Spezifikationen haben keinen Anspruch auf vollständige Überprüfung der Korrektheit des Codes aber sie helfen dabei, Fehler frühzeitig zu vermeiden [12]. Da Spezifikationssprachen eine mathematisch präzise Notation für die Aufzeichnung bestimmter Programmeigenschaften sind, kann mit ihnen der Programmcode besser gestaltet werden [12]. Formale Spezifikationen können für viele Teile einer Softwareentwicklung eingesetzt werden, wie die Anforderungsspezifikation, die Entwicklung der Softwarearchitektur, den Programmcode oder diverse Tests [12].

Die verschiedenen Spezifikationen können je nach Art der Artefakte oder Art der Eigenschaft, die sie beschreiben in unterschiedliche Klassen eingeteilt werden [12]. Wenn Spezifikationen bezogen auf ihre Eigenschaften eingeteilt werden kann man folgende Gruppen von BISLs unterscheiden:

Die erste Gruppe sind die Lebendigkeitseigenschaften (liveness properties), welche besagen, dass möglicherweise etwas Positives passieren wird, wie eine Antwort auf eine Systemabfrage [12]. Das Gegenteil dazu bilden die Sicherheitseigenschaften (security properties), welche aussagen, dass nichts Negatives passiert, wie beispielsweise die Absicherung eines Systems vor Abstürzen [12]. Die Sicherheitseigenschaften bestehen dabei aus den Eigenschaften des funktionalen Verhaltens und den Ressourceneigenschaften [12].

Die Eigenschaften des funktionalen Verhaltens (functional behavior properties), welche Systemoperation oder Zustandsänderungen aufzeigen konzentrieren sich hauptsächlich auf die Beziehung zwischen Vor- und Nachbedingungen [12]. Die zweite Unterklasse der Sicherheitseigenschaften sind die Ressourceneigenschaften (resource properties). Diese beziehen sich auf Einschränkungen der Ressourcen, wie beispielsweise Zeit oder Speicherplatz und bestimmen, wieviel von einer Ressource durch eine Operation verwendet werden darf [12]. Die letzte Klasse sind die zeitlichen Eigenschaften (temporal properties), welche die Beziehung zwischen Systemereignissen und Zustandsübergängen beschreiben [12].

Wenn man die Spezifikation nach Art der Artefakte einteilt kann man die vier folgenden Kategorien erkennen (siehe Abbildung 1):

Abbildung 1: *Übersicht der Artefakt-basierten Zuordnung von BISL*

Die erste Kategorie umfasst die Spezifikationen auf Analyselevel (analysis-level specifications), welche Domänenmodelle innerhalb einer Domäne behandeln [12]. Dazu gehören BISL wie zum Beispiel Alloy, Z oder die AsmL (Abstract State Machine Language) [12]. Spezifikationssprachen auf Anforderungslevel (requirements-level specifications), wie RSML (Requirements State Machine Language) und Teile von UML beschreiben das Verhalten eines Systems als Ganzes [12]. Die Entwicklung der Softwarearchitektur kann mit Hilfe der architektonischen Spezifikationen (architectural specifications), welche die beabsichtigten Verbindungen zwischen den Komponenten eines Programms erläutern, unterstützt werden [12]. Die letzte Klasse zu denen Sprachen, wie Spec#, Larch oder die JML gehören, sind die Spezifikationen der Code-Schnittstellen (Code interface specification) [12]. Sprachen für die funktionellen Eigenschaften verlassen sich dabei stark auf Behauptungen, Vor- und Nachbedingungen sowie Invarianten und sie beschreiben Verträge zwischen Client und Anbieter nach dem Design by Contract-Ansatz [12].

Durch die genannten Kategorisierungen wurden die unterschiedlichen BISL schon vorklassifiziert. Im nächsten Abschnitt wird sich mit der Kategorisierung der JML innerhalb der Spezifikationssprachen der Kategorie Code-Schnittstellen Spezifikation genauer auseinandergesetzt.

IV. Kategorisierung der BISL

Die unterschiedlichen BISLs der Kategorie Code-Schnittstellen werden hier in drei Unterkategorien eingeteilt: programmierspezifischen, programmiersprachen-offenen BISLs und Spezifikationssprachen, die eine eigene Programmiersprache sind.

Zu den programmiersprachenspezifischen BISLs gehören neben JML auch die Spezifikationssprache Spec#. Sie wurde von Microsoft Research entwickelt und gilt als Erweiterung zur Programmiersprache C# [14]. Diese wird durch Vor- und Nachbedingungen, Invarianten und Non-Null-Types ergänzt [14]. Sie wurde maßgeblich von Eiffel, AsmL und JML beeinflusst [14]. Der Vorteil von Spec# ist, dass es ein eigenes System für die Entwicklung mit Spezifikationen bereitstellt [14]. Dabei nutzt es alle Datentypen von C# [14], was unter Umständen negative Auswirkungen haben kann, da die Komplexität erhöht wird und dadurch weitere Fehler entstehen können.

Weitere programmiersprachenspezifische BISLs stellen Vienna Development Method (VDM) und dessen Erweiterungen dar [15]. VDM wurde von IBM entwickelt und nutzt die Programmiersprache PL/I [15]. Weiterhin kann das Tool von VDM für Java und C++ Programme eingesetzt werden [16]. Im Gegensatz zu Spec# und JML basiert VDM nicht auf dem Eiffel Ansatz sondern auf der Z-Notation und auf der Vienna Development Language (VDL) [12, 17]. Durch den Aufbau von VDM ergeben sich verschiedene Versionen, welche unterschiedliche Notationen nutzen. Das Erlernen von VDM ist einfach und günstig, wodurch der Schulungsaufwand für Mitarbeiter gering bleibt. Weiterhin gilt VDM als abgeschlossen, weshalb keine Änderungen in der Syntax erwartet werden [15].

ANNA ist eine weitere programmiersprachenspezifische BISL, welche mit der Programmiersprache Ada arbeitet [17]. Diese Programmiersprache beeinflusste die Entwicklung von ANNA maßgeblich, wodurch den Ada-Programmierern der Einstieg in die Spezifikationssprache leicht fiel [17]. Heute wird ANNA nicht mehr genutzt, da es veraltet ist und durch SPARK abgelöst wurde.

Alle vorgestellten programmiersprachenspezifischen BISLs werden in Tabelle 1 gegenübergestellt. Sie haben jedoch alle denselben Nachteil: Sie sind nur auf ihre jeweilige Programmiersprache anwendbar. Ein Vorteil entsteht durch den möglichen Einsatz und die Kopplung verschiedenster Tools.

Unter den Spezifikationssprachen, die eine eigene Programmiersprache sind finden sich Sprachen, wie Eiffel, SPARK oder Gypsy. Eiffel wurde von Bertrand Meyer entwickelt und ist eine objektorientierte Programmiersprache, mit welcher der Design by Contract-Ansatz entwickelt wurde [18]. Zu dieser Kategorie gehört auch das ebengenannte SPARK, welches auf Ada und Eiffel basiert und eine eigene Programmiersprache darstellt [19]. Mit Hilfe von SPARK können lauffähige Programme erstellt werden, jedoch können ebenfalls Ada- und C-Programme mit SPARK kombiniert werden. Eine weitere Spezifikationssprache der Kategorie stellt das auf Pascal basierende Gypsy dar [20].

	JML	Spec#	VDM /VDM-SL, VDM++, VDM-RT, VDM-10	ANNA
Entwickelt von	Gary Leavens [9]	Microsoft Research [14]	IBM [15]	B.Krieg-Brueckner and 0. Owe [24]
Programmier-sprachen	Java [1]	C# [14]	PL/I [15]	Ada [17]
Allgemeines	Nutzt Java Syntax [11]	Erweiterung zu C# [14]	• VDMTool kann auch für Java und C++ genutzt werden [16] • Verschiedene Versionen	• Erweitert Ada [17] • Abgelöst von SPARK
Vorteile	siehe Fazit	eigenes System für die Entwicklung mit Spec# wird bereitgestellt [14]	• keine Änderungen in der Syntax erwartet [15] • Erlernen von VDM ist einfach und günstig [15]	ist für Ada User einfach zu lernen und zu nutzen [17]
Nachteile	siehe Fazit	Testgenerierung ist komplex, da alle Datentypen von C# genutzt werden [14]	Unterschiede der Notationen	veraltet und wird nicht mehr genutzt
Beeinflusst von	Eiffel, Larch, refinement calculus [1]	JML, AsmL und Eiffel [14]	VDL, in Z definiert [12,17]	Ada [17]

Tabelle 1: *Vergleich programmiersprachenspezifischer BISL*

Die letzte Gruppe sind die programmiersprachen-offenen BISLs. Darunter fallen Sprachen, wie Z, OCL, Larch und die RESOLVE family. Z wurde von der Universität Oxford in Zusammenarbeit mit industriellen Partnern, wie IBM und Inmos entwickelt [6]. Die Z-Notation dient zur mathematischen Beschreibung von Systemen und wurde durch mehrere Organisationen standardisiert [6]. OCL beschreibt die Randbedingungen eines UML-Modells und wurde von IBM entwickelt [21]. Operatoren und Ausdrücke, welche aus Vor- und Nachbedingungen für Methoden bestehen, können durch OCL effektfrei spezifiziert werden [21]. Larch wurde am MIT Laboratory for Computer Science und dem DEC Systems Research Center entwickelt [22]. Es kann in zwei Schichten aufgeteilt werden, wobei eine Schicht die zugrundeliegenden Abstraktionen enthält und die andere Schicht die Larch Shared Language darstellt [22]. Diese Schicht ist die gemeinsame Sprache, welche Abstraktionen verwendet [22]. Larch kann so für verschiedene Programmiersprachen eingesetzt werden, was auch bei der RESOLVE family möglich ist.

RESOLVE ist eine integrierte Sprache, welche mathematische Spezifikationen und imperatives Programmieren kombiniert, um so eine volle funktionale Verifikation der Programmmodule zu ermöglichen [23].

Um die JML in die anderen BISLs einzuordnen, werden die vorher beschriebenen Kategorien genutzt. JML gehört zu den Code-Schnittstellen Spezifikationen und wird dort in die programmiersprachenspezifischen BISL eingeordnet. Viele Sprachen nutzen Eiffel, wie zum Beispiel JML, OCL und Spec#. Aufgrund dessen kann man Eiffel als übergeordnete BISL einstufen. JML kann im Gegensatz zu Larch oder OCL nicht für andere Programmiersprachen genutzt werden und ist somit auch diesen untergeordnet. Bei der Entwicklung mit verschiedenen Programmiersprachen kann die JML somit nicht oder nur eingeschränkt genutzt werden.

Die Spezifikationssprache OCL basiert ebenfalls auf dem Design by Contract-Ansatz, jedoch ist sie gegenüber JML programmiersprachen-offen. Somit ist OCL allgemeingültiger als JML. Jedoch ist der große Vorteil von programmiersprachenspezifischen BISLs, dass sie die Zugriffsmodifikatoren, wie static, public oder private der jeweiligen Sprache nutzen können. Durch die Nutzung der Zugriffsmodifikatoren können detailliertere Designentscheidungen und eine höhere Ausdruckskraft zur Beschreibung von Programmverhalten erreicht werden [9]. Aufgrund dessen bietet sich die Nutzung von JML an, da sie im Gegensatz zu programmiersprachen-offenen BISLs, Modifikatoren nutzen kann [9], welche die Beschreibung von Java-Programmen verbessert. Dies liegt daran, dass OCL eine generische Spezifikationssprache ist und die spezifischen Ausdrücke der jeweiligen Programmiersprachen nicht nutzen kann [9]. Spezifikationssprachen, die auf eine Programmiersprache zugeschnitten sind, sind vor allem in den späteren Entwicklungsphasen für detaillierte Designentscheidungen besser geeignet als eine generische BISL [9].

Der Einsatz programmiersprachen-offener BISLs sollte deshalb gut überlegt werden, da hiermit keine komplette Abdeckung für die Überprüfung gewährleistet werden kann.

V. Die Nutzung in Unternehmen

Die Einführung von Spezifikationssprachen in den Entwicklungsprozess erzeugt zunächst Kosten in Form von Mitarbeiterschulungen und möglichen organisatorischen Maßnahmen. Jedoch werden sich diese Kosten bald relativieren, da funktionale Fehler während der Entwicklung vermieden werden können. Es wird daher weniger Zeit für die Fehlerbeseitigung nach der Veröffentlichung in Anspruch genommen und die Mitarbeiter in der Qualitätssicherung werden entlastet. Die Vielzahl an Werkzeugen, welche JML nutzen, bietet einen einfachen und kostengünstigen Einstieg in die Spezifikationssprache JML. Dadurch ist es möglich, ausgefeiltere Techniken der Nutzung von Tools zu entwickeln bis eine gute Kombination von Tools und Techniken gefunden wurde, die kosteneffektiv ist und für alle Entwicklungen des Unternehmens eingesetzt werden kann. Durch den Einsatz des Tools jmlunit erhalten Mitarbeiter eines Unternehmens eine gute Unterstützung bei Unit-Tests, um das Testen von Java-Klassen und -Interfaces weitestgehend zu automatisieren und somit Ressourcen einzusparen. Das genannte Tool generiert für die Tests eine Unit-Testklasse, welche auf dem Runtime

Assertion Checker (ebenfalls ein Werkzeug basierend auf JML) beruht [9]. Die Testklasse sendet Nachrichten an die Objekte der Java-Klassen, während der Testcode Assertion-Fehler von den Methodenaufrufen auffängt und entscheidet, ob die Testdaten die Vorbedingungen erfüllt haben oder nicht [9]. Solche Assertion-Fehlschläge legen nicht den Fehlschlag des gesamten Tests fest, sondern deuten auf ungenügende und somit falsche Implementierung in Bezug auf die Spezifikation hin [9]. Trotz dieses Werkzeugs ist der Nutzer immer noch für die Erstellung der Testdaten zuständig. Durch die bestehende Testklasse wird es dem Nutzer vereinfacht Testdaten einzugeben [9]. Zudem hat der Nutzer bei Bedarf die Möglichkeit, zusätzliche selbstgeschriebene JUnit-Tests einzufügen [9]. Mit jmlunit kann somit Unit-Test mit minimalem Programmieraufwand betrieben werden, bei dem viele Fehler in der Implementierung und der Spezifikation entdeckt werden können. Der Aufwand die Testfälle zu schreiben ist dann ähnlich dem Aufwand der Entwicklung der Spezifikationen [9]. Weitere Tools existieren um die Erstellung von Dokumentationen zu unterstützen. Durch diese können ebenfalls Ressourcen eingespart werden.

Allgemein ist die Nutzung von JML empfehlenswert, da es erhebliche Vorteile im Bereich der Qualitätssicherung bietet. Die JML kommt hauptsächlich im Bereich der agilen Entwicklung zum Einsatz. Hiermit kann die Rollenverteilung und die Schnittstellenspezifikation für alle Beteiligten klar definiert werden. Durch die Festlegung der Spezifikationen wird im Voraus eine gute Kommunikationsgrundlage zwischen den Entwicklerteams gesichert und Abweichungen von der Spezifikation während der Implementierung vermieden. In der agilen Softwareentwicklung kann JML in den Entwicklungsphasen Implementierung und Test genutzt werden. Es dient der Fehlervermeidung und trägt zum besseren Verständnis des Codes unter den einzelnen Entwicklerteams bei. Anforderungen und Verantwortlichkeiten werden dadurch klar definiert und das beabsichtigte Programmverhalten zugesichert.

VI. Fazit

Durch fehlerhafte oder schlechte Software geht viel Zeit verloren, da viele Fehler in der Wartung behoben und ausgebessert werden müssen. Durch JML können diese Fehler frühzeitig vermieden werden. Im Gegensatz zu der allgemeinen Kritik an formalen Methoden, dass sie schwer verständlich seinen und wenig genutzt werden [8], ist die JML für Java Programmierer bedingt durch die Java-ähnliche Syntax schnell zu erlernen und so auch einfach zu nutzen. Ein Vorteil ist dabei, dass sich die Annotationen und Kommentare nahtlos in den neuen oder den bestehenden Code eingliedern, wodurch eine detaillierte Beschreibung des Programmdesigns möglich wird. Jedoch wird der Code durch die zusätzlichen Teile unübersichtlicher und dadurch schwieriger zu lesen. Die Lesbarkeit kann jedoch durch die aspektorientierte Programmierung mit Hilfe von AspectJML verbessert werden. Weiterhin steht eine große Anzahl an Verifikationswerkzeugen, welche dieselbe Notation nutzen, für das Spezifizieren des Programverhaltens über JML zur Verfügung. JML deckt weitaus mehr Funktionalitäten ab, als von den bisherigen Tools genutzt wird [11]. Die Sprache ist immer noch in der Entwicklung, weshalb nicht nur die Toolunterstützung nicht komplett ausgebaut ist, sondern auch das Benutzerhandbuch Lücken aufweist. Das Handbuch ist noch nicht vollständig beschrieben und bisher wenig aussagekräftig. Weiterhin besteht noch keine eindeutige und offizielle Definition von JML [11].

Eine weitere Hürde in der Nutzung von JML stellt die schlechte Einbindung in die Entwicklungsumgebung Eclipse dar, welche den Einstieg in JML erschwert.

Die JML bietet trotz der noch andauernden Entwicklung einen guten Einstieg in die Thematik der Spezifikationssprachen [11]. Im Gegensatz zu anderen Behavioral Interface Specification Languages muss JML noch an der Definition und der Abdeckung der Funktionalität in den einzelnen Tools arbeiten. Spec# dient hier als Vorreiter, da es schon eine formale Referenzsemantik hat und die Toolunterstützung ausgereift ist. Eine weitere Herausforderung für die JML ist die stetige Anpassung an die jeweilige Java Version [11].

Letztlich sind die Arbeiten an JML noch nicht abgeschlossen und bieten noch viel Raum für Verbesserungen und Optimierungen an. Ein standardisiertes Vorgehen für formale Spezifikationen gestaltet sich wegen nicht eindeutiger Definition und individuellen Tools als schwierig. Jedoch ist JML so stabil, dass es für den industriellen Einsatz genutzt werden und maßgeblich zur Verbesserung des Codes beitragen kann.

Referenzen

[1] Lavens, Gary T. / Baker, Albert L. et al. (2006): "*Preliminary Design of JML:A Behavioral Interface Specification Language for Java*". ACM SIGSOFT Software Engineering Notes, Ausgabe 31, Nummer 3.

[2] Leavens, Gary T. (2007): "*Tutorial on JML, the Java Modeling Language*". ASE'07. 5–9, November 2007. Atlanta,Georgia. USA.

[3] Bender, Frederik (2003): "*Proseminar - Der Airbusabsturz LH2904 in Warschau*". http://www4.in.tum.de/lehre/seminare/ps/WS0203/desaster/Bender-Airbus-Ausarbeitung-13-01-03.pdf, abgerufen am 6. Mai 2015.

[4] Durumeric, Zakir/ Kasten, James et al. (2014): "*The Matter of Heartbleed*". IMC'14. 5–7, November 2014. Vancouver. Canada.

[5] Stelzer Dirk (2000): "*Qualitätsmanagement in der Softwareentwicklung*". Computer Reseller News, Nummer 13, Seiten 50-54.

[6] Zeller, Andres (2005): "*Formale Spezifikation mit Z*". Universität des Saarlandes, Saarbrücken. https://www.st.cs.uni-saarland.de/edu/se/2005/zed.pdf, abgerufen am 8. Juni 2015.

[7] Leavens, Gary T. / Cheon, Yoonsik (2006): "*Design by Contract with JML*". Iowa State University. http://www.eecs.ucf.edu/~leavens/JML/jmldbc.pdf, 6. Mai 2015.

[8] Schneider, Kurt (2012): "*Abenteuer Softwarequalität - Grundlagen und Verfahrenfür Qualitätssicherung und Qualitätsmanagement*". dpunkt-Verlag, 2. Auflage.

[9] Burdy, Lilian / Cheon, Yoonsik et al. (2003): "*An overview of JML tools and applications*". Electronic Notes in Theoretical Computer Science, Ausgabe 80, Seiten 75–91.

[10] He, Jifeng/ Liu, Zhiming (2006): "*Formal Methods and Software Engineering - 8th International Conference on Formal Engineering Methods*", ICFEM 2006 Macao,China, November 2006, Proceedings. Springer-Verlag.

[11] Bruns, Daniel (2011): "*Eine formale Semantik für die Java Modeling Language*". Springer-Verlag. Informatik-Spektrum, Februar 2012, Ausgabe 35, Auflage 1, Seiten 45-49.

[12] Hatcliff, John / Leavens, Gary T. et al. (2010): "*Behavioral Interface Specification Language*". ACM Computing Surveys (CSUR). Ausgabe 4, Nummer 3, Juni 2012, Artikel 16. New York, USA.

[13] Chalin,Patrice (2004): "*JML Support for Primitive Arbitrary Precision Numeric Types: Definition and Semantics*". Journal Of Object Technology, Ausgabe 3, Nummer 6, ETH Zurich.

[14] Barnett, Mike / Leino, K. Rustan M. et al. (2004): "*The Spec# Programming System: An Overview*". Microsoft Research, Redmond, WA, USA. CASSIS 2004.

[15] Plat, Nico / Larsen, Peter Gorm (1992): "*An Overview of the ISO/VDM-SL Standard*". ACM SIGPLAN Notices, Band 27, Ausgabe 8, August 1992.

[16] Fitzgerald , John / Larsen, Peter Gorm (2008): "*VDMTools: advances in support for formal modeling in VDM*". ACM SIGPLAN Notices, Band 43, Ausgabe 2, Februar 2008, Seiten 3-11, ACM New York, USA.

[17] Krieg-Brückner, Bernd / Luckham, David C. (1980): "*ANNA : Towards a Language for Annotating Ada Programs*". SIGPLAN '80 Proceedings of the ACM-SIGPLAN symposium on Ada programming language, Seiten 128 - 138, ACM, New York, USA.

[18] Wilder, Dan (1995): "*Introduction to Eiffel*". Linux Journal , Band 1995, Ausgabe 14, Specialized Systems Consultants, Inc. (hrsg). ACM.

[19] Barthe,Gilles/ Burdy, Lilian (1973): "*Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*". Springer-Verlag.

[20] Ambler, Allen L./ Good, Donald I. (1977): "*Gypsy: A language for specification and implementation of verifiable programs*". Proceedings of an ACM conference on Language design for reliable software, ACM New York, USA.

[21] ISO. (2012): "*Information technology - Object Management Group Object Constraint Language (OCL)*". edition standard: ISO/IEC 19507.

[22] Wing , Jeannette M. (1987): "*Writing Larch Interface Language Specifications*". ACM Transactions on Programming Languages and Systems, Band 9, Ausgabe 1, Januar 1987, Seiten 1-24.

[23] Welch, Daniel / Cook, Charles et al. (2014): "*A Web-Integrated Verifying Compiler for RESOLVE: A Research Perspective*". ISEC' 14, Februar 2014. Chennai, India. ACM.

[24] Luckham, David C. / von Henke, Friedrich W. (1985): "*An Overview of Anna, a Specification Language for Ada*". Stanfort University . IEEE SOFTWARE.

Einfluss von aspektorienterter Programmierung auf die Laufzeit

Aiko Isselhard
Informatik Master
Hochschule Furtwangen
aiko.isselhard@hs-furtwangen.de

Nico Konop
Informatik Master
Hochschule Furtwangen
n.konop@hs-furtwangen.de

***Abstract*— Aspektorientierte Programmierung stellt eine einfache Möglichkeit Cross-Cutting Concerns, die sich über eine Softwareanwendung verteilen, in wenige konkrete Aspekte zusammenzufassen. Das integrieren der Aspekte geschieht durch Veränderung des Kontrollflusses der Anwendung. Diese Arbeit untersucht den Einfluss dieser Kontrollflussänderung auf die Laufzeitperformance anhand einer rechenintensiven Methode. Bei den Performancetests wurden die Frameworks AspectJ und PostSharp verwendet.**

I. Einleitung

In diesem Beitrag wird der Einfluss von aspektorientierter Programmierung (AOP) auf die Laufzeit untersucht. AOP wird weder in Java noch in .Net, beziehungsweise C# nativ unterstützt. Um AOP zu ermöglichen, werden Frameworks von Drittanbietern benötigt. Die Wahl dieser Frameworks hat einen erheblichen Einfluss auf das Testergebniss. Hier werden die Frameworks AspectJ (Java) und PostSharp (.Net) betrachtet; sie haben einen ähnlichen Funktionsumfang und erfüllen folgende Kriterien:

- Native Implementierung: Jedes Framework ist in der jeweiligen Sprache (Java, bzw. C#) programmiert
- „State of the Art“: Die Frameworks entsprechen dem Quasi-Standard für AOP in Java und AOP in .Net
- Die Frameworks sind weitgehend anerkannt und haben eine breite Nutzer-Basis
- Beide Frameworks haben eine gewisse „Reife“ erreicht
- Beide Frameworks bieten kommerzielle Unterstützung: Das sichert ihre Weiterentwicklung und macht diese Frameworks auch für kommerzielle Projekte interessant.
- Beide Frameworks werden aktiv gewartet und weiterentwickelt

Alle Vergleiche sind dadurch zwangsläufig ein Vergleich der Frameworks. Ein reiner Vergleich von Java und .Net ist aufgrund der fehlenden nativen Unterstützung nicht möglich.

Obwohl beide Produkte demselben Prinzip folgen, gibt es in der Implementierung und in der Funktionalität doch Unterschiede. Ein Teil dieser Unterschiede ist auf sprachabhängige Unterschiede zurück zu führen, ein anderer Teil auf die nicht eindeutige Definition von AOP. Die Literatur findet zwar einen gemeinsamen Grundkonsens, lässt jedoch gewissen Interpretationsspielraum wenn es in die Details geht. So ist beispielsweise genau definiert was ein „Join Point“ (siehe Kapitel II) ist, jedoch nicht wo „Join Points“ auftreten dürfen.

II. Grundlagen

AOP ist ein relativ junges Konzept: Es wurde erstmals von Gregor Kiczales am Xerox PARC-Institut (mittlerweile bekannt als PARC, a Xerox Company) vorgestellt. Die aspektorientierte Programmierung folgt denselben Grundprinzipien wie die objektorientierte Programmierung (OOP) und die objektorientierte Analyse und Design (OOAD): Kapselung, Abstrahierung Wiederverwendbarkeit, Separation of Concerns (SoC) und Single Responsible Principle (SRP)[1]. AOP ermöglicht es, diese Prinzipien auch in Bereichen einzuhalten, in denen reine OOP dazu nicht mehr in der Lage ist: bei den sogenannten Cross-Cutting-Concerns, zu Deutsch die Querschnittsfunktionen. Cross-Cutting-Concerns sind Anforderungen, die quer über viele Systeme gelten. Beispiele sind: Logging, Transaktionen oder Rechtemanagement.

Abbildung 1 und Abbildung 2 zeigen eine Auflistung aller Module des Apache HTTP Server nach Lines of Codes (LoC). In Abbildung 1 wurde das XML-Parser-Modul markiert. Gemäß der SoC und SRP ist das XML-Parsing in ein Modul gekapselt und es bestehen keine Abhängigkeiten zu anderen Modulen. In Abbildung 2 wurden alle Aufrufe an das Logging-Framework markiert. Hier ist sehr gut zu sehen, dass das Logging stark mit allen anderen Modulen verwoben ist. Das Prinzip der SoC und der SRP ist mit traditionellen OOP/OOAD-Mitteln nicht einzuhalten.

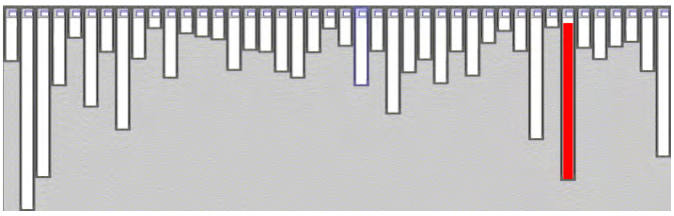


Abbildung 1. Apache XML Parser [2]

In dem Logging-Beispiel könnte ein solcher Aspekt folgendermaßen lauten: „Logge den Funktionsnamen, wenn vorhanden die Parameter-Namen und ihren Wert sowie die Aufrufzeit. Anschließend soll die Laufzeit der Funktion gemessen und ausgegeben werden.“. Um diese Anforderung ohne AOP zu

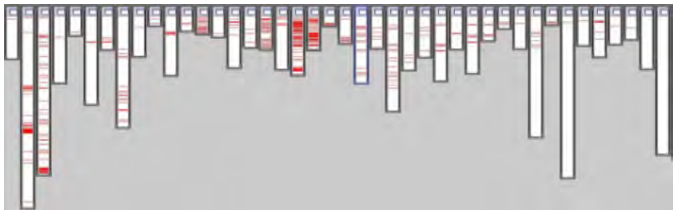


Abbildung 2. Apache Logging [2]

lösen, müsste zu Beginn des Funktionsaufrufs der Name und die Parameter geloggt werden und die Startzeit gespeichert werden. Am Ende der Funktion müsste dann die Laufzeit berechnet werden. Das führt zu viel ähnlichem Code, der per Copy und Paste in jede Funktion eingefügt wird. Der für die eigentliche Aufgabe der Funktion nicht benötigte Code verschlechtert sowohl die Lesbarkeit, als auch die Wartbarkeit. Wird der Funktionsname oder werden die Parameter geändert, muss immer auch der entsprechende Logging-Code angepasst werden.

AOP kapselt die Cross-Cutting-Concerns in separate Aspekte. Die funktionale Beschreibung des Aspekts wird Advice genannt. Der Advice beschreibt demnach, was der Aspekt tut. In diesem Fall würde der Advice den Funktionsnamen mit den Parametern loggen, bevor die Funktion ausgeführt wird. Diesen Punkt nennt man Pointcut. In der aspektorientierten Programmierung werden alle Punkte, an denen ein Aspekt eingreifen kann Join Points genannt. Mögliche Join Points sind beispielsweise: Vor und nach einem Funktionsaufruf, das werfen einer Exception oder das erstellen eines neuen Objekts. Die Punkte an denen der Aspekt dann wirklich greift sind die Pointcuts. In dem Beispiel werden die Join Points des Logging-Aspekts vor und nach dem Methodenaufruf.

Das AOP-Framework fügt alle Advices an den entsprechenden Pointcuts ein. Dieser Prozess wird weaving, auf Deutsch „verweben“ genannt. Es gibt zwei unterschiedliche Weaving-Strategien: Compile-Time- und Runtime-Weaving [3]. Beim Runtime-Weaving werden die Advices erst zur Laufzeit in die Pointcuts eingesetzt. Das heißt die ausgeführte Bytecode wird zur Laufzeit verändert. Runtime-Weaving wird von PostSharp, der AOP-Implementierung in .Net, nicht unterstützt (siehe Kapitel IV). Beim Post-Compile-Time-Weaving werden die Aspekte zur Kompilierungszeit in den Bytecode integriert. Das Weaving kann entweder Pre- oder Post-Compile erfolgen. Das Pre-Post-Compile-Time-Weaving kommt dem Fällen Copy-Paste aus dem OOP-Beispiel sehr nahe. Bevor der Quellcode in Maschinen- bzw. Bytecode übersetzt wird, werden alle Advices textuell an die Pointcuts kopiert. Beim Post-Post-Compile-Time-Weaving wird der Quellcode zuerst in Maschinen- bzw. Bytecode übersetzt. Das AOP-Framework bearbeitet und verändert dann den resultieren Bytecode und integriert die Advices in diesen. AOP-Frameworks die diese Methode nutzen, sind deshalb meisten nicht mit anderen Tools, die den Bytecode ändern, z.B. Obfuscatoren, kompatibel. Die Laufzeitunterschiede des Advisecodes nach verändern des Kontrollflusses, sind zwischen Runtime und Compile Weaving nicht signifikant. Runtime Weaving verursacht jedoch Performanceeinbußen beim dynamischen setzen der Pointcuts [4]. Sowohl AspectJ, als auch PostSharp unterstützen Post-Post-Compile-Time-Weaving.

III. AspectJ

AspectJ ist eine unter der Eclipse Foundation geführtes Open Source-Erweiterung zur Umsetzung von aspektorientierten Projekten in Java. Nachfolgende Sprachkonstrukte und Beschreibungen richten sich nach Version 1.8 des AspectJ-Compilers [1] und Version 2.2.4 der AspectJ Development Tools (AJDT) [2]. AspectJ ist sowohl für nicht kommerzielle als auch kommerzielle Projekte frei im Rahmen der Eclipse Public Licence 1.0 [3] verfügbar.

Die AspectJ Development Tools intergrieren sich als Plugin in die Entwicklungsumgebung Eclipse. Nach der Integration werden zusätzlich zu den bereits bestehenden Projekten, AspectJ Projekte zur Verfügung gestellt.

Das Plugin übernimmt dabei die Konfiguration und die Auswahl des AspectJ-Compilers. Ein Projekt ist somit nach der Installation des Plugins übersetz- und ausführbar.

A. Aufbau/Arten von Aspekten

Der strukturelle Aufbau und die Organisation von Aspekten orientieren sich stark an Java Klassen. Jeder Aspekt wird analog zu Klassen in Java in einer separaten Datei mit der Dateiendung „.as“ gespeichert. Wie in Listing 1 zu sehen, werden zunächst ein oder mehrere Pointcuts deklariert. Der erste Teil deklariert dabei den Verweis auf den eigentlichen Advice Code. Die zweite Hälfte der Pointcutdeklaration definiert die Bedingung unter welcher der Joinpoint greift. AspectJ erlaubt es, mehrere Bedingungen mit den binären logischen Operatoren Und und Oder zu verknüpfen oder mithilfe des unären Nicht-Operators explizit ausschließen.

```
1 public aspect AroundAspect {
2   pointcut fibTest() :
3   call(long Around.fib(long));
4    long around() : fibTest() {
5     ...
6     return ...;
7   }
8 }
```

Listing 1. Around Aspect

Die Advicedeklaration ist vergleichbar mit einer Java-Methodendeklaration. Dabei wird zwischen drei Arten von Advices unterschieden:

- before
- after
- around

Alle drei Advicetypen bestehen grundsätzlich aus einer Advicetypendeklaration und einem eindeutigen methodenkonformen, parameterlosen Bezeichner. Dieser wird ebenfalls bei der Pointcutdeklaration verwendet wird.

Der Before-Advice greift vor der enstprechenden Methode, der After-Advice danach. Zusätzlich kann der After-Advice dazu verwendet werden, bei erfolgreichem Aufruf einer Funktion ein Rückgabeobjekt abzufangen, indem das Schlüsselwort

[1] http://www.eclipse.org/aspectj/
[2] http://www.eclipse.org/ajdt/
[3] http://eclipse.org/legal/epl-v10.html

„returning“ und einer Objektdeklaration als Parameter angegeben wird. Des Weiteren ist es Möglich, mit dem Schlüsselwort „throwing“ auf eine Ausnahme zu reagieren.

Ein Around-Aspekt ersetzt im Gegensatz zum Before- und After-Aspekt die gesamte Methode durch den Advice. Der ursprüngliche Code wird in eine neu generierte Methode ausgelagert. Dadurch muss beim Around-Advice zusätzlich ein Rückgabetyp angegeben werden, welcher dem der Originalmethode entspricht. Der Zugriff auf die Originalmethode innerhalb des Advice erfolgt durch den Methodenaufruf „proceed()“.

Um auf kontextspezifische Informationen eines Pointcuts zugreifen zu können, wurde in AspectJ Version 1.1 das Schlüsselwort „thisJoinPoint“ eingeführt. Dieses ermöglicht es zum Beispiel auf Parameter eines Methodenzugriffes wie in Listing 2 zuzugreifen.

```
before() : fibTest() {
  Object[] args = thisJoinPoint.getArgs();
  ...
}
```

Listing 2. Parameterzugriff bei AspectJ

Die Methode getArgs() gibt die Parameter der vom Aspekt betroffenen Methode, als ein Array von Objektinstanzen zurück. Bei primitiven Datentypen wird automatisch ein Boxing durchgeführt.

B. Instanzierungsverhalten von Aspekten

Wie in Tabelle I beschrieben, gibt es vier unterschiedliche Arten wann ein Aspekt unter AspectJ instanziert werden kann. Jeder Aspekt wird im Standardfall einmal pro Anwendung instanziert. Die Instanzierungsart kann durch Angabe eines zusätzlichen Schlüsselwortes bei der Aspektdeklaration beeinflusst werden.

Tabelle I. INSTANZIERUNGSARTEN

AspectJ Bezeichnung	Instanzierungsart	Beispielaufruf
singelton	Singelton	MyAspect.aspect()
perthis	Per Objekt	MyAspect.aspect(targetObject)
pertarget	Per Typ	MyAspect.aspect(Target.class)
perflow/perflowbelow	Per Kontrollfluss	MyAspect.aspect()

C. Weaving

AspectJ untestützt drei Weaving Arten:

- Post-Compile-Time-Weaving
- Post Compile Weaving
- Load Time Weaving (auch Runtime Weaving genannt)

In diesem Beitrag wird auschließlich (Pre-)Post-Compile-Time-Weaving betrachtet. Innerhalb des AspectJ-Frameworks wird beim Post-Compile-Time-Weaving der eigentlichen Weavingvorgang als Preprozess vor dem Kompilieren als Teilprozess des AspectJ-Compilers durchgeführt. Die Eingabedateien in Form von Java Klassen und Aspekten, werden in jeweilige .class-Dateien übersetzt. Klassen und Schnittstellen welche von Aspekten nicht betroffen sind, werden beim Weavingprozess übersprungen und direkt vom Compiler in Bytecode übersetzt [5].

D. Weaving-Ablauf im Detail

Advicedeklarationen werden vom AspectJ-Compiler in Standard Java Methoden übersetzt. Before und After Advicedeklarationen werden als statische Methodenaufrufe an der entrechenden Stelle des Pointcuts eingefügt. Um zu gewährleisten, dass der After Advice auch beim Auftreten einer Ausnahme ausgeführt wird, wird der ursprüngliche Code zusätzlich von einem try-catch Block umrahmt.

Around Advices unterscheiden sich von Before und After Advices, indem sie den ursprünglichen Code eines Pointcuts in eine seperate Methode verschieben. Diese enthält neben den Parametern der Originalmethode zuzätzliche Kontextinformationen wie aktuelle Aspektinstanz und Informationen zur Methodensignatur der Quellmethode. Die ursprüngliche Methode wird durch den Aufruf der generierten Advicemethode ersetzt [6].

IV. POSTSHARP

PostSharp ist eine kommerzielle AOP-Implementierung für .Net. PostSharp bietet auch eine Freeware-Variante an, die in der Funktionalität beschnitten ist. Viele Optimierungsmöglichkeiten und die Möglichkeit, Einfluss auf das Weaving zu nehmen, sind ausschließlich in der Kommerziellen Variante möglich. In diesem Beitrag wird die kommerzielle Version PostSharp[4] in der Version 4.1.14 getestet. PostSharp besteht zum einen aus der PostSharp-Bibliothek, sowie dem PostSharp-AddIn für Microsofts Visual Studio (Hier wird Microsoft Visual Studio Ultimate 2013[5] Version 12.0.31101.00 Update 4 verwendet). Die PostSharp-Bibliothek ermöglicht es, Aspekte als Klassen zu entwickeln und bietet unterstützende Hilfsfunktionen, mit denen der Build-Prozess beeinflusst werden kann. Das PostSharp-AddIn greift automatisch in den Build-Prozess des Visual-Studios ein und führt das Post-Post-Compile-Time-Weaving durch. Durch PostSharp wird AOP so in die IDE integriert, dass der Build/Deployment-Prozess für den Nutzer transparent ist.

PostSharp unterstützt nur Post-Compile-Time-Weaving. Bibliotheken und Executables müssen in .NET oft mit einem „starken Namen“ versehen werden. Eine Bibliothek kann beispielsweise nur im Windows Global Assembly Cache (GAC) platziert werden, wenn diese einen „starken Namen“ hat. In diesem Prozess wird die Bibliothek mit einer GUID (Globally Unique Identifier) versehen und signiert. Bibliotheken mit „starkem Namen“ sind auch Versionssicher. Die Bibliothek kann nur geladen werden, wenn die Signierung, die GUID und die Version übereinstimmen. Da Runtime-Weaving den Bytecode nach dem signieren ändern würde, wäre die Bibliothek oder die Executable in vielen Bereichen nicht mehr nutzbar.

In PostSharp entspricht ein Aspekt einer normalen Klasse, die von einem bestimmten Basis-Typ erbt: MethodInterceptionAspect, OnMethodBoundaryAspect und LocationInterceptionAspect. Jede dieser Aspekt-Basisklasse bietet unterschiedliche Join Points. Das hat den Vorteil, dass Aspekte, vererbt werden können. So ist es möglich mithilfe von Vererbungshierarchien komplexe Frameworks zu entwickeln.

[4] https://www.postsharp.net/
[5] https://www.visualstudio.com/

Alle Aspekte müssen mit dem „Serializable“-Attribut versehen werden. Dieses Attribut sorgt dafür, dass die Klasse in einen Bytestream serialisiert und aus diesem wieder instanziiert werden kann. Dabei werden auch private Variablen und Zustände serialisiert. Postsharp instantiiert alle benötigten Aspekte während dem Weaving und legt den serialisierten Bytestream in die Assembly. Dadurch muss der Aspekt nicht mehr zur Laufzeit erstellt werden. Wird der Aspekt benötigt, muss lediglich der Bytestream in den RAM kopiert werden. Das hat auch zur Auswirkung, dass der Konstruktor nur zur Kompilierungszeit aufgerufen wird. Für eine Initialisierung zur Laufzeit werden vom PostSharp-Framework entsprechende Methoden angeboten.

In der Standard-Konfiguration, wird pro Klasse ein Aspekt instanziiert. Das bedeutet alle Objekte derselben Klasse greifen auf dieselbe Instanz des Aspekts zu. Es gibt auch die Möglichkeit, einen Instanz pro Objekt zu nutzen. Die zusätzliche Instanziierung der Aspekte hat negative Auswirkungen auf die Laufzeit. In diesem Beitrag werden ausschließlich Aspekte genutzt, deren Instanz zwischen allen Objekten der selben Klasse geteilt wird.

PostSharp bietet zwei Möglichkeiten um die Pointcuts zu definieren:

1) **[Externe Konfiguration]** über eine externe Konfiguration-Datei können Regeln definiert werden, die Aspekten bestimme Pointcuts zuweisen. Die Regeln reichen von „Wende Aspekt A an der Methode B der Klasse C an“, bis zu komplexen Regeln mit regulären Ausdrücken. So ist es zum Beispiel möglich, einen Aspekt auf alle Methoden von Klassen anzuwenden, deren Klassen-Namen mit „Entity“ enden, außer der Methoden-Name entspricht dem Klassen-Namen. Mit dieser Regel würde der Aspekt auf alle Methoden der Entities, außer dem Konstruktor angewendet.
2) **[Inline Konfiguration]** Der Aspekt kann als Attribut an jede Klasse oder Methode zugewiesen werden. Mit dieser Methode ist im Code jederzeit ersichtlich, welcher Aspekt auf welche Methode angewandt wird. Wird das Attribut einer Methode zugewiesen, wird der Aspekt nur auf diese Methode angewendet. Wird das Attribut einer Klasse zugewiesen, wird der Aspekt auf alle Methoden der Klasse angewendet.

Dabei können Abhängigkeiten zwischen den Aspekten definiert werden, so dass z.B. Aspekt A nur angewendet werden kann, wenn Aspekt B angewendet wird. über die Abhängigkeiten kann auch die Reihenfolge, in der die Aspekte ausgeführt werden, definiert werden.

A. *LocationInterceptionAspect*

Der LocationInterceptionAspect ermöglicht ein Eingreifen bei Properties. Dieser Aspekt-Typ bietet zwei Join Points: OnSetValue und OnGetValue. Die Join Points sind einem Proxy änlich und leiten den Kompletten Aufruf um. Das heißt der Aufruf des Getters, bzw. des Setters kann umgeleitet oder komplett verboten werden. Da Java keine Properties kennt, wird dieser Aspekt-Typ im Weiteren nicht näher untersucht.

B. *MethodInterceptionAspect*

Der MethodInterception-Aspekt hat, verglichen mit den anderen PostSharp-Aspekten, den größten Einfluss auf die Performance. Er funktioniert wie ein Proxy, der vor einen Funktionsaufruf geschaltet wird. Der usprüngliche Aufruf wird komplett umgeleitet.

In Listing 3 wird jeder Aufruf an eine Methode, die mit diesem Aspekt versehen ist, an die OnInvoke-Methode umgeleitet. Der Aspect kann die Funktion mit „args.Proceed“ aufrufen. Er kann wie in Listing 1 die Methode beliebig oft ausführen oder den Aufruf an eine andere Funktion weiterleiten. Da die Methode immer im OnInvoke-Kontext ausgeführt wird, müssen alle Parameter, sowie Informationen der Methode in den Stack der OnInvoke-Methode kopiert werden. Dabei werden alle Wert-Typen (Eng.: Value-Types) automatisch geboxt.

```
1 [Serializable] public sealed class Aspect:
2   MethodInterceptionAspect {
3   public override void OnInvoke(
4     MethodInterceptionArgs args) {
5     for (var i=0; i<3; ++i)
6     args.Proceed();
7    }
8 }
```

Listing 3. MethodInterceptionAspect

C. *OnMethodBoundaryAspect*

Der MethodBoundary-Aspekt bietet die meisten Joint-Points der drei vorgestellten Aspekte: OnEntry, OnSuccess, OnException und OnExit. Die Funktionsweise dieses Aspekt lässt sich, wie in Abbildung 3 dargestellt, am Besten an einem aus Java oder C# bekannten Try-Catch-Finally-Block demonstrieren.

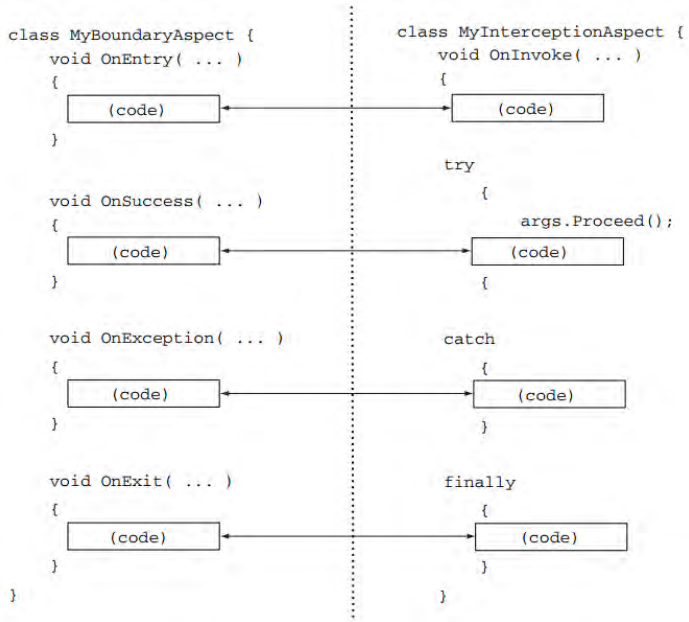


Abbildung 3. Pointcuts von MyBoundaryAspect im Vergleich zu MyInterceptionAspect [7, S. 83]

Um den Aspekt ohne PostSharp zu simulieren, muss der Funktionscode, in diesem Fall „args.Proceed()“, von einem

Try-Catch-Finally-Block umrahmt werden. OnEntry wird vor dem Try-Block, direkt nach dem Eintreten in die Funktion ausgeführt. OnExit wird im Finally-Block aufgerufen, dadurch wird Die Abarbeitung des OnExit-Advices im Finally-Block durchgeführt. Dadurch wird sichergestellt, dass dies Unabhängig vom Auftreten einer Exception geschieht. Wird der Funktionscode ohne eine Exception ausgeführt, kann die OnSuccess-Methode aufgerufen werden. Andernfalls wird die Exception im Catch-Block gefangen und OnException ausgeführt. Der Unterschied zu dem Interception-Aspekt besteht darin, dass der Funktionscode im Kontext der ursprünglichen Funktion ausgeführt wird.

Ein Blick in den dekompilierten Code zeigt, dass PostSharp den Funktionscode in einem Try-Catch-Finally Block umrahmt. PostSharp erzeugt dabei nur die Blöcke, die auch verwendet werden. Wird beispielsweise der OnExit-Join Point nicht genutzt, wird kein Finally-Block erzeugt. Wird nur der OnEntry-Join Point benötigt, fügt PostSharp keinen Try-Catch-Block ein. Duch das weglassen nicht benötigter Try-, Catch- oder Finally-Blöcke wird die Laufzeit-Performance optimiert.

Auch beim MethodBoundary-Aspekt müssen alle Parameter und Methoden-Informationen gesammelt und an den Aspekt übergeben werden. Das führt, wie beim Interception-Aspekt zu einem boxing aller Wert-Typen (Eng.: Value-Types). Allerdings sammelt PostSharp diese Informationen nur, wenn sie auch verwendet werden. Verwendet ein Aspekt die Parameter nicht, werden diese auch nicht kopiert und dadurch nicht geboxt.

V. Versuchsaufbau

Für den Performance-Test wurden verschiedene Szenarien entwickelt. Jedes Szenario wurde in Java und in .Net/C# einmal mit und einmal ohne das ensprechende AOP-Framework implementiert. Die Variante ohne AOP-Framework wurde mit nativen Mitteln der jeweiligen Sprache entwickelt, so dass sie dieselbe Funktionalität abdeckt. Der Boundary-Aspect (PostSharp)/Around(AspectJ) wurde durch einen Try-Catch-Finally-Block und der Interception-Aspect durch eine Wrapper-Funktion ersetzt. Dabei sollte es für den Nutzer transparent sein, welche Implementierung mit oder ohne AOP-Framework genutzt wird.

Die Tests wurden wie in Tabelle II auf zwei Test-Rechnern ausgeführt.

Tabelle II. Versuchsumgebungen

Hardwareumgebung 1	**Hardwareumgebung 2**
Windows 8.1 Pro x64 Intel(R) Core(TM) i7-3520@2.90GHz 16,0 GB RAM	Windows 8.1 Pro x64 Intel(R) Core(TM) i5-4200M@2.50-GHz 8,00 GBRAM

Für die Tests wurden folgende Tools und Frameworks genutzt:

- PostSharp 4.1.14
- Microsoft Visual Studio Ultimate 2013 Version 12.0.31101.00 Update 4
- Microsoft .NET Framework Version 4.5.51650 x64
- JetBrains ReSharper Ultimate 2015.1 9.1.1
- Version: Luna Service Release 2 (4.4.2) Build id: 20150219-0600
- AspectJ Version: 1.8.5.20150128171000
- Eclipse AspectJ Development Tools 2.2.4.e44x-20150202-1700

Jeder Performance-Test wurde mit an einer eigenen Klasse mit einer Methode ausgeführt. Die Methode berechnet die Fibonacci-Folge rekursiv. Durch die Rekursion wird auch jeder Pointcut entsprechend mehrfach ausgeführt: fib(1)=0, 1 Funktionsaufruf; fib(2)=1, 2 Funktionsaufrufe; fib(3)=2, 3 Funktionsaufrufe; fib(4)=3, 9 Funktionsaufrufe; fib(5)=5, 15 Funktionsaufrufe; fib(6)=8, 25 Funktionsaufrufe; fib(7)=13, 41 Funktionsaufrufe; fib(8)=21, 67 Funktionsaufrufe; fib(9)=34, 109 Funktionsaufrufe; fib(10)=55, 177 Funktionsaufrufe.

Für den Test wurde die Zeit gemessen, die benötigt wurde um die Funktion 30000 mal aufzurufen. Vor und nach jeder Zeitmessung wurde jeweils 0,5 Sekunden gewartet. Damit wird der Testumgebung Zeit gegeben eventuelle Hintergrundaufgaben, wie z.B. Garbage-Collection oder andere Störfaktoren, durchzuführen. Dieser Prozess wurde insgesamt 300 mal durchgeführt und alle Zwischenergebnisse wurden notiert.

Die Versuchsrechner wurden vom Internet getrennt und alle nicht benötigten Hintergrundprozesse deaktiviert, um potenzielle Störfaktoren während der Testläufe zu beseitigen. Trotzdem kann eine vollständige Isolation nicht gewährleistet werden.

VI. Testbeschreibung und Auswertung

A. Before- und After-Test

Testbeschreibung:

Beim Before- und After-Test wird der Advice jeweils beim Betreten und Verlassen der Fibonacci-Funktion ausgeführt. Dazu wurden vier Testreihen ausgeführt, deren Ergebnisse in Abbildung 4 dargestellt sind.

In den ersten zwei Testreihen wurde der Overhead eines leeren Pointcuts getestet. Zu diesem Zweck wurde ein Aspekt mit leerem Advice für die Join Points Before und After entwickelt. Dieser wurde sowohl an einer Objekt-Methode [HU1 Empty, HU2 Empty] und als auch an einer statischen Methode [HU1 Empty Static, HU2 Empty Static] getestet.

Ziel der dritten und der vierten Testreihe ist es, den Overhead der beim Sammeln der Funktionsparameter anfällt, zu messen. Dazu wurde im Advice die Fibonacci-Funktion vom Parameterwert der Originalfunktion n - 1 ausgerechnet (siehe Listing 4 und Listing 5). Dabei wurde die Fibonacci-Funktion im Advice getrennt implementiert. Diese Testreihe wurde ebenfalls auf Objekt-Methoden [HU1, HU2] und auf statischen Methoden [HU1 Static, HU2 Static] getestet.

```
1 before() : fibTest() {
2   Object[] args = thisJoinPoint.getArgs();
3   fib(((Long) args[0]) - 1);
4 }
```

Listing 4. Advice mit fib(n-1) in Java)

```
public override void OnEntry(MethodExecutionArgs
    args) {
  Fib(((long) args.Arguments[0]) - 1);
}
```

Listing 5. Advice mit fib(n-1) in C#

Um den Aspekt ohne AOP zu simulieren, wurde der Advice in eine eigene Klasse ausgelagert. Die Originalfunktion ruft auf einem Objekt dieser Klasse die Before- und After-Funktion vor beziehungsweise nach der Fibonacci-Berechnung auf. Dabei wurde die After-Funktion im simulierten Aspekt nicht im Finally-Block aufgerufen.

```
BeforeAndAfter aspect = new BeforeAndAfter();
new Fib(aspect).fib(8);
```

Listing 6. Dependency Injection

Eine Instanz des simulierte Aspekts wurde bei der Instanziierung der Fibonacci-Klasse übergeben (siehe Listing 6).

Auswertung:

In dem Versuch wurden die vier Testreihen jeweils mit AOP und mit ihrem OOP-Äquivalent 30000 mal ausgeführt. Die Zeit für 30000 Funktionsaufrufe wurde gemessen und über 300 Wiederholungen gemittelt. Abbildung 4 zeigt den prozentualen Laufzeitunterschied zwischen der AOP-Testreihe und der Testreihe von ihrem jeweiligen OOP Äquivalent.

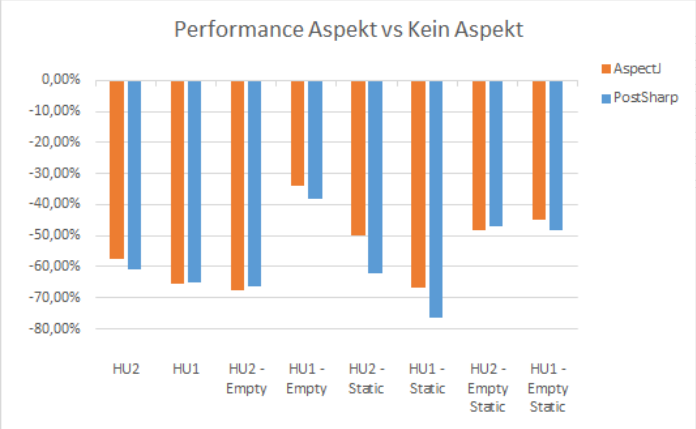


Abbildung 4. Before & After - Test, Perfomance Aspekt vs Kein Aspekt

Wie in Abbildung 4 zu sehen ist, wirkt sich AOP in allen Testreihen signifikant auf die Laufzeit aus. Die leeren Testreihen zeigen, dass der Aufruf eines Advices im Pointcut bis zu 70% langsamer als ein konventioneller Methodenaufruf ist. Das Sammeln der Funktionsparameter und das boxen dieser hat keinen messbaren Einfluss auf die Laufzeit. Die Auswertung zeigt, dass obwohl die statischen Versuchsreihen absolut gesehen kürzere Laufzeiten als die nicht-statischen haben, die prozentualen Performanceeinbußen nicht signifikant geringer ausfallen.

B. *try-catch-finally - Test*

Testbeschreibung:

Beim Try-Catch-Finally-Test wird der Advice jeweils beim Betreten, beim Auftreten einer Ausnahme, nach dem erfolgreichen ausführen und beim Verlassen der Fibonacci-Funktion ausgeführt. Insgesamt werden pro Methodenaufruf drei Pointcuts ausgeführt. Dazu wurden vier Testreihen ausgeführt, deren Ergebnisse in Abbildung 4 dargestellt sind.

Wie im Before- und After-Test, wurden leere Advices auf Objekt-Methoden [HU1 Empty, HU2 Empty] und auf statischen Methoden [HU1 Empty Static, HU2 Empty Static] ausgeführt. Neben den leeren Testreihen wurde die Fibonacci-Function auch mit dem Parameter der Originalfunktion n - 1 berechnet (Objekt-Methode: HU1, HU2; Statische Methode: HU1 Static, HU2 Static).

Im Gegensatz zum vorherigen Test, wurde in diesem der durch den eingeführten Try-Catch-Finally-Block erzeugte Overhead gemessen.

Um diesen Aspekt ohne AOP zu simulieren, wurde der Advice in eine eigene Klasse ausgelagert. Der ursprüngliche Code der Fibonacci-Funktion wurde in einen Try-Block geschachtelt (siehe Listing 7). Die Pointcuts wurden wie in Abbildung 3 gezeigt, aufgerufen.

```
public long Fib(long n) {
  _aspect.OnEntry(n);
  long res = 0;
  try {
    if (n <= 1) res = n;
    else res = Fib(n - 1) + Fib(n - 2);
    _aspect.OnSuccess(n);
  }
  catch {
    _aspect.OnException(n);
  }
  finally {
    _aspect.OnExit(n);
  }
  return res;
}
```

Listing 7. Fibonacci im Simulierten Aspekt

Auswertung:

In dem Versuch wurden die vier Testreihen jeweils mit AOP und mit ihrem OOP-Äquivalent 30000 mal ausgeführt. Die Zeit für 30000 Funktionsaufrufe wurde gemessen und über 300 Wiederholungen gemittelt. Abbildung 5 zeigt den prozentualen Laufzeitunterschied zwischen der AOP-Testreihe und der Testreihe von ihrem jeweiligen OOP-Äquivalent.

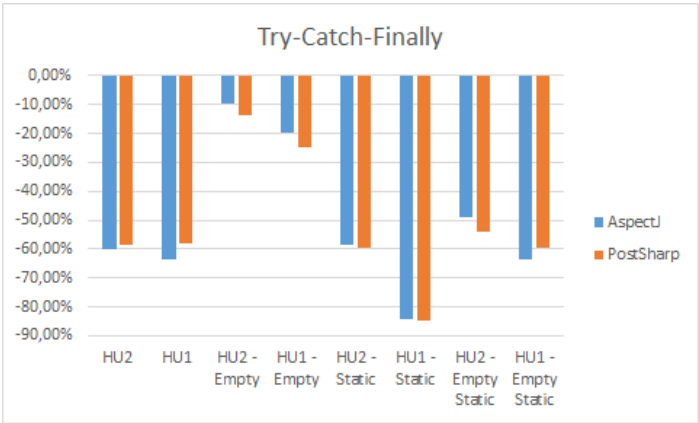


Abbildung 5. try-catch-finally - Test, Perfomance Aspekt vs Kein Aspekt

Diese Testreihe bestätigt das Ergebnis der Before und After Testreihe, dass die Advice-Aufrufe bis zu 70% langsamer sind als konventionelle Methoden-Aufrufe. Eine Ausnahme bilden hier die Empty-Testreihen (HU1 Empty, HU2 Empty). Ein Vergleich des erzeugten Bytecodes zeigt, dass die Implementierung, abgesehen von der Initialisierung des AOP-Frameworks, nahezu identisch sind. Das erklärt einen Performance-Verlust von nur 10-20%. Dieses Ergebnis gilt nicht für die statischen Testreihen, die bis zu 60% Performance kosten. Auch für Aspekte, die auf statische Methoden wirken, muss eine Instanz des jeweiligen Aspekts erzeugt werden. In den Pointcuts an der statischen Methode erfolgt ein Aufruf an die jeweilige Methode der Aspekt-Instanz. Diese Methode ist nicht statisch. Dieser Umstand macht die Performance-Vorteile zunichte, die der Aufruf einer statischen Methoden verglichen mit einer Objekt-Methode mit sich bringt. Dieser Umstand führt ferner dazu dass die Versuchsreihen [HU1 Empty Static, HU2 Empty Static] bis zu 60% Performanceeinbußen kosten und die Testreihen [HU1 Empty, HU2 Empty] auf 20% Performanceeinbußen kommen.

C. Around - Test

Testbeschreibung:

Beim Around-Test wird der Funktionsaufruf umgeleitet und der Advice führt die Fibonacci-Funktion zweimal aus (siehe Listing 8 und Listing 9). Der Around-Aspekt entspricht somit einem der Funktion vorgeschaltetem Proxy.

```
long around() : fibTest() {
  long result=0;
  for (int i = 0; i < 2; i++) {
    result = proceed();
  }
  return result;
}
```

Listing 8. Around Aspekt in Java

```
public override void OnInvoke(
  MethodInterceptionArgs args) {
  for (var i = 0; i < 2; ++i) {
    args.Proceed();
  }
}
```

Listing 9. Around Aspekt in PostSharp

Um einen den Around-Aspekt ohne AOP zu simulieren, wurde der Advice in eine eigene Klasse ausgelagert. Dem Advice wird beim Aufruf ein Funktionspointer, beziehungsweise in Java ein Interface übergeben. Der Funktionspointer und das Interface wird von der Advicefunktion zweimal aufgerufen. Der Mock wird der Fibonacci-Berechnung beim Instanziieren übergeben (siehe Listing 11).

Die fib-Funktion musste dahingehend geändert werden, dass sie den injizierten Advice aufruft. Die beiden Lösungen unterscheiden sich im Ausführungskontext der Fibonacci-Funktion. In der Lösung mit AOP, wird der Advice im Kontext der aufrufenden Funktion und die Fibonacci-Funktion im Kontext des Advices ausgeführt. In der simulierten Implementierung wird der Advice im Kontext der Fibonacci-Funktion und die Fibonacci-Funktion im Kontext der aufrufenden Funktion ausgeführt (siehe Listing 10). Der in Listing 10 und 11 aufgeführte Quelltext ist in Java und C# identisch.

```
public long fib(long n) {
  return aspect.onInvoke(() -> {
    if (n <= 1) return n;
    return fib(n - 1) + fib(n - 2);
  });
}
```

Listing 10. Methodenaufruf über Anonyme Methode

```
public long OnInvoke(Func<long> func) {
  long res = 0;
  for (var i = 0; i < 2; ++i) {
    res = func();
  }
  return res;
}
```

Listing 11. Simulierter Around-Aspekt

Im Gegensatz zu den vorherigen Tests, wurden in diesem Test nur zwei Testreihen, an einer Objekt-Methode und an einer statischen Methode ausgeführt.

Auswertung:

In dem Versuch wurden die vier Testreihen jeweils mit AOP und mit ihrem OOP-Äquivalent 30000 mal ausgeführt. Die Zeit für 30000 Funktionsaufrufe wurde gemessen und über 300 Wiederholungen gemittelt. Abbildung 5 zeigt den prozentualen Laufzeitunterschied zwischen der AOP-Testreihe und der Testreihe von ihrem jeweiligen OOP-Äquivalent.

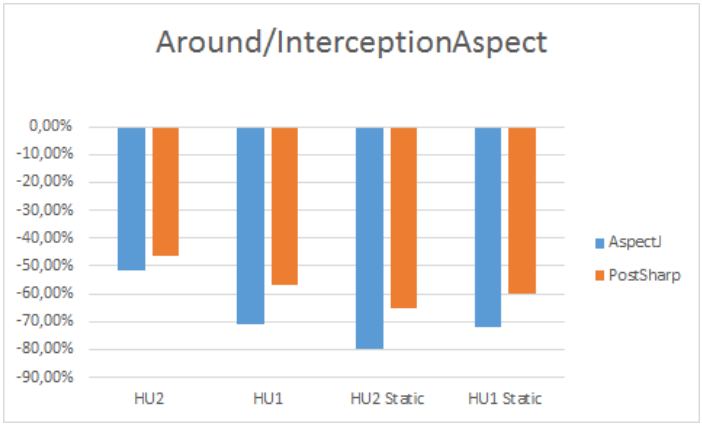


Abbildung 6. Around - Test, Perfomance Aspekt vs Kein Aspekt

Dieser Aspekt ist im Vergleich zu den vorherigen Aspekten der aufwendigste. Er hat den größten Einfluss auf die absolute Laufzeitperformance. Dies ist zurückzuführen auf den in der Eingangsbeschreibung erwähnten Kontextwechsel und das Ersetzen aller Referenzen auf die Fibonacci-Funktion. Allerdings ist auch die Simulation im Vergleich aufwendiger, so dass im Mittel ein Laufzeit-Performanceverlust von 60% entsteht.

VII. Fazit

In diesem Beitrag wurde zuerst auf die Unterschiede zwischen PostSharp und AspektJ eingegangen. Anhand von exemplarischen Aspekten an der Fibonacci-Funktion, wurde der Einfluss der Aspekt-Orientierten-Programmierung auf die Laufzeit-Performance untersucht. Die Testergebnisse zeigen dass AOP einen deutlichen Einfluss von bis 70% gegenüber

konventionellem OOP hat. Allerdings wurde der reine Overhead der Advice-Aufrufe gemessen. Verglichen mit der Laufzeitperformance einer vollständigen Software, ist der Overhead vernachlässigbar. Der Anteil eines Methodenaufrufs an der Laufzeitperformance ist so gering, dass er selbst wenn er um 70% erhöht wird, vernachlässigbar klein ist.

Nicht betrachtet wurde der mögliche Laufzeit-Performancegewinn durch die neuen Möglichkeiten die AOP im Software-Design bietet. AOP ist wie der Schritt von der Prozeduralen Programmierung in die OOP. OOP bringt auf den ersten Blick viel Overhead mit, kann aber durch intelligentes Software-Design zu leistungsfähigeren Programmen führen.

LITERATUR

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *ECOOP'97 - Object-Oriented Programming*, ser. Lecture Notes in Computer Science, M. Akşit and S. Matsuoka, Eds. Springer Berlin Heidelberg, 1997, vol. 1241, pp. 220–242.

[2] [Online]. Available: http://www.denimgroup.com/know_artic_aspectoriented.html

[3] F. Schmied and A. Cyment, "Aspect-oriented weaving and the .net common language runtime," *Software, IET*, vol. 1, no. 6, pp. 251–262, December 2007.

[4] D. Ansaloni, W. Binder, P. Moret, and A. Villazon, "Dynamic aspect-oriented programming in java: The hotwave experience," in *Transactions on Aspect-Oriented Software Development IX*, ser. Lecture Notes in Computer Science, G. Leavens, S. Chiba, M. Haupt, K. Ostermann, and E. Wohlstadter, Eds. Springer Berlin Heidelberg, 2012, vol. 7271, pp. 92–122.

[5] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold, "An overview of aspectj," in *ECOOP 2001 - Object-Oriented Programming*, ser. Lecture Notes in Computer Science, J. Knudsen, Ed. Springer Berlin Heidelberg, 2001, vol. 2072, pp. 327–354.

[6] E. Hilsdale and J. Hugunin, "Advice weaving in aspectj," in *Proceedings of the 3rd International Conference on Aspect-oriented Software Development*, ser. AOSD '04. New York, NY, USA: ACM, 2004, pp. 26–35.

[7] M. D. Groves, *AOP in .NET: Practical Aspect-Oriented Programming*. Manning Publications, 2013. [Online]. Available: "http://www.manning.com/groves/"

Verwendung von aspektorientierter Programmierung zur Vereinfachung von entfernten Methodenaufrufen mit Apache Thrift

Stefan Retzlaff, Serbest Hammade
Hochschule Furtwangen
Fakultät Informatik
Furtwangen, Deutschland
Email: {stefan.retzlaff, serbest.hammade}@hs-furtwangen.de

***Zusammenfassung*—Diese Ausarbeitung beschreibt eine Möglichkeit wie die Verteilung von entfernten Methodenaufrufen mit Hilfe der aspektorientierten Programmierung umgesetzt werden kann. Es wird aufgezeigt wie dieses Programmier-Paradigma unter Verwendung von Apache Thrift zu stark vereinfachter Integration von verteilten Methodenaufrufen eingesetzt werden kann. Durch die Verwendung von benutzerdefinierten Annotationen im Zusammenhang mit der aspektorientierten Verteilung erhält der Entwickler ein hohes Maß an Flexibilität und eine Übersicht über die neu hinzugefügten Funktionen.**

***Keywords*—AOP, Thrift, Verteilte Systeme**

I. Einleitung

Das Einbinden dieser entfernten Methodenaufrufe in eine Anwendung erfordert üblicherweise die Änderung des Programms an vielen verschiedenen Stellen, da häufig nicht nur eine einzelne Methode ausgelagert werden soll. Diesen Änderungen folgt meist noch die Anpassung der Unit-Tests.
Diese Ausarbeitung stellt ein Lösungskonzept vor, mit dem die Methodenverteilung realisiert werden kann, ohne den Code ändern zu müssen. Es wird aufgezeigt wie das Konzept der aspektorientierten Programmierung(AOP) verwendet werden kann um die Querschnittsfunktion der Methodenverteilung zu lösen. Weiterhin wird ein Weg vorgestellt, wie die Entwickler mit wenig Aufwand und in sehr übersichtlicher Art und Weise diese Methodenverteilung in ein bestehendes System integrieren und bei Bedarf wieder deaktivieren können. Dabei wird der Sourcecode durch die Separation der Funktionen übersichtlicher und wiederverwendbarer im Vergleich zu der Standardlösung der Methodenverteilung. Durch die Verwendung von Apache Thrift ist diese Lösung zusätzlich noch plattformunabhängig, was den Einsatz in einer hetrogenen Systemumgebung vereinfacht.
Mit der Verwendung von aspektorientierter Programmierung in einem Projekt erhöht sich auch die Komplexität des Entwicklungsprozesses, da eine neue Technologie eingesetzt wird, welche deutlich anders anzuwenden ist als die übliche objektorientierte Entwicklung. Allerdings wird in dieser Arbeit gezeigt wie AOP die Komplexität des Quellcodes selbst verringern kann in dem die Querschnittsfunktionen von den Kernfunktionen separiert werden.
In Abschnitt II wird das Konzept der aspektorientierten Programmierung vorgestellt. Abschnitt III gibt einen kurzen Einblick in Verteilte Systeme und das Framework Thrift, welches in dieser Arbeit verwendet wird. Anschließend wird AOP und Thrift zusammen in dem Lösungsansatz verwendet, welcher in Abschnitt IV beschrieben wird. In Abschnitt V werden die Themen beschrieben, welche in dieser Arbeit nicht in ausreichender Tiefe betrachtet werden konnten. Diese bieten in der Zukunft noch Potential für weitere Studien. In Abschnitt VI werden zwei Arbeiten zu ähnlichen Themen aufgeführt und der Unterschied zu dieser Arbeit erläutert und anschließend in Abschnitt VII die Arbeit mit einem Fazit abgerundet.

II. Aspektorientierte Programmierung

Die objektorientierte Programmierung (OOP) hat sich als grundlegendes und erfolgreiches Programmierparadigma seit Jahren etabliert. Es gibt allerdings auch hier Fragen denen sich der Entwickler stellen, z.B. der Einbindung von Funktionen, die in dem gesamten Programm verwendet werden sollen. Diese werden als Querschnittsfunktionen(Crosscutting Concerns) bezeichnet.[2] Im Jahr 1996/1997 stellen Kiczales u. a. das Konzept der aspektorientierten Programmierung vor und entwickeln wenige Jahre später AspectJ, die erste praktische Implementierung dieses Konzeptes.[2, 3]

A. Grundlegendes zu AOP

Wie bereits kurz erwähnt, werden mit der aspektorientierten Programmierung Anforderungen umgesetzt, die in mehreren Teilen des objektorientierten Codes eingebunden werden müssen. Dies führt zu Abhängigkeiten der Klassen und Methoden auf diese Querschnittsfunktionen. Diese Abhängigkeiten sorgen beispielsweise für Probleme, wenn es um die Erstellung von Unit-Tests geht, da hier nur die Funktion einer Komponente getestet werden soll und nicht die Zusammenarbeit der Komponente mit anderen Komponenten. In dieser Arbeit wird der Crosscutting Concern der verteilten Methodenverteilung aufgegriffen. Methoden, welche sich beispielsweise als Ressourcenintensiv herausstellen, sollen auf einem entfernten System ausgeführt werden. Dies sollte idealerweise möglich sein, ohne das der Code der Kernfunktionen stark verändert werden muss. In folgenden werden die Techniken vorgestellt, um diese Methodenverteilung mittels AOP implementieren zu können.

1) Join Points: Innerhalb eines objektorientierten Projekts gibt es viele Stellen, an denen die Ausführung einer Querschnittsfunktion einsetzen könnte. Dies Punkte werden als

Join Points bezeichnet. Beispiele für solche Join Points finden sich im Ablauf eines Programms, bevor oder nachdem eine Methode aufgerufen wird.

2) Pointcuts: Die Querschnittsfunktionen sollen nicht an jedem möglichen Join Point einsetzen, sondern nur an einem oder mehreren bestimmten, genau definierten Join Point. Ein solcher, definierter Join Point wird als Pointcut bezeichnet.
Welche Join Points verwendet werden können und wie diese deklariert werden, richtet sich nach der verwendeten Implementation des AOP-Ansatzes. In dieser Arbeit wird die Sprache AspectJ (siehe auch Abschnitt II-B) verwendet und die eingesetzten Join Points beziehen sich auf die Aufrufe von Methoden. Die Methoden können beispielsweise anhand des Namens, der Übergabewerte oder der Rückgabewerte identifiziert werden.[3]
Zusätzlich können auf die Methode bezogene Annotationen zur Identifizierung dieser Methode verwendet werden. In der hier vorgestellten Lösung werden zwei neue Annotationen (Custom Annotations) erstellt um die Methoden zu identifizieren. Diese Variante bietet dem Entwickler die Möglichkeit sich innerhalb des Quellcodes seiner Anwendung für die Ausführung von Aspekten zu entscheiden. Die zusätzlichen Annotationen werden dabei ausschließlich von dem aspektorientierten Code verwendet. Einen weiteren Vorteil bringt diese Variante bei der Codegenerierung, wie in Abschnitt IV näher erläutert wird.

3) Advice: Das Advice bezeichnet die Anweisung, welche an dem Join Point ausgeführt werden soll, der durch den entsprechenden Pointcut identifiziert wurde. AspectJ bietet die Möglichkeit ein Advice vor (before), nach (after) oder stattdessen/während (around) einer identifizierten Methode auszuführen.[3]
Der Advice-Code kann auf die Übergabeparameter der Methode und auf Informationen der zu der Methode gehörenden Annotationen zugreifen. Wird ein Around Advice spezifiziert, so muss innerhalb des Advice die Methode ausgeführt werden und der Rückgabewert zurückgegeben werden, denn dieses Advice wird anstatt der Methode ausgeführt. Auf diese Weise kann die aspektorientierte Programmierung verwendet werden, um den Inhalt einer Methode zu ersetzen. Diese Funktion wird in dieser Arbeit verwendet, um die Ausführung einer Methode durch einen entfernten Methodenaufruf zu ersetzen um anschließend das Ergebnis wieder zurückzugeben.

4) Aspekte: Ein Aspekt ist eine Sammlung von zusammengehörigen Advices und Pointcuts, und hat damit innerhalb der AOP eine ähnliche Rolle wie eine Klasse in der OOP. Zusätzlich können Aspekte, wie auch eine objektorientierte Klasse, noch Variablen und Methoden beinhalten.

5) Weaving: Als Weaving wird der Prozess des Einflechtens des aspektorientierten Codes in den objektorientierten Code bezeichnet. Dies geschieht beim Kompilieren des Programms. Es werden die Advices an den Join Points, welche durch die Pointcuts identifiziert wurden, eingebunden. Das Ergebnis dieses Prozesses ist, wie auch bei dem normalen Kompilieren, das ausführbare Programm, mit dem Unterschied das dieses Programm um den AOP Code erweitert wurde.[3]

B. AspectJ

AspectJ ist eine Implementierung des AOP-Konzeptes für die Programmiersprache Java. Es ist eine Erweiterung der Java Sprache und somit sind alle gültigen Java Programme auch gültige AspectJ Programme. Die mit AspectJ erzeugten Programme müssen auf allen Plattformen, auf denen eine Java Virtual Machine (JVM) installiert ist, funktionieren. Daher sind die vom AspectJ-Compiler erzeugten Klassen und Methoden vollständig kompatibel zu dem Java Standard.[3]

C. Entwicklungsprozess

Der AspectJ-Compiler ajc bietet nach [8] drei Möglichkeiten für den Zeitpunkt des Weavings an. Je nach Einführungszeitpunkt von AOP in ein Projekt bieten sich verschiedene Verfahren an:

1) Compile-time: Der AspectJ-Compiler kompiliert die Aspekte direkt mit dem auf die Pointcuts im OOP-Code anzuwendenden Änderungen. Dieses Verfahren eignet sich vor allem bei Projekten, in denen AOP von Anfang an eingesetzt wird.[8]

2) Post-compile time: Wie in der Abbildung 1 abgebildet, injiziert der AspectJ-Compiler die Aspekt-Anweisungen in bereits kompilierte Klassen (Java-Byte-Code) und kann somit auf bestehende Projekte und in Bibliotheken angewendet werden.[8]

3) Load-time: Während des Ladevorganges von Klassen injiziert der AspectJ-Compiler Aspekt-Anweisungen in den Java-Byte-Code. Dies erfordert die Auslieferung der AspectJ-Bibliothek an die Benutzer.[8]
Der Java-Byte-Code kann in den Verfahren Post-compile time und Load-time als class-Dateien oder als jar-Datei vorliegen. In der hier vorgestellten Lösung wird das Post-compile time-Verfahren angewendet, weiteres dazu im Abschnitt IV.

III. Verteilte Systeme

Ein verteiltes System ist eine Ansammlung von eigenständigen Computern, die dem Benutzer als eine Einheit erscheinen. Das Hauptziel solcher verteilten Systeme ist das zur Verfügung stellen von Ressourcen und diese kontrolliert und effizient mit anderen zu teilen. Diese Ressourcen können beispielsweise Drucker, Speicherkapazität, Daten oder andere Netzwerkressourcen sein. Es gibt verschiedene Gründe diese Ressourcen mit anderen System teilen zu wollen. Die

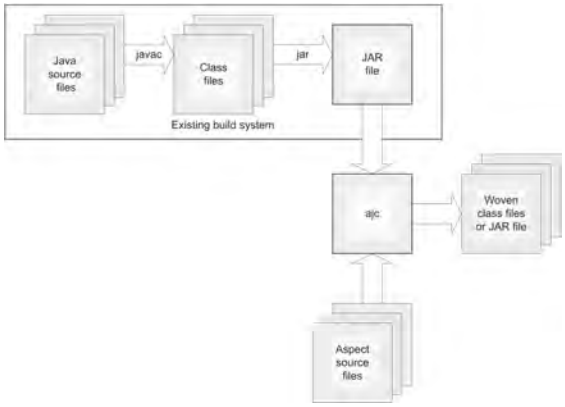


Abbildung 1. Aspekte in bestehende Build-Systeme einweben[3]

häufigsten Gründe sind die Wirtschaftlichkeit, die Skalierbarkeit, die Verfügbarkeit sowie die Plattformunabhängigkeit. Dagegen spricht die hohe Komplexität eines verteilten Systems, dessen Fehleranfälligkeit und der administrative Mehraufwand. Einen Drucker mit anderen zu teilen ist in der Anschaffung günstiger, als wenn jedes System seinen eigenen Drucker besitzt, allerdings können dabei Ressourcenkonflikte entstehen, wenn mehrere Benutzer gleichzeitig diesen Drucker bedienen wollen. Die Entwicklung und der Einsatz von Verteilten Systemen begründen sich nicht allein aus der Machbarkeit, vielmehr müssen die Anforderungen an ein System den Aufwand rechtfertigen.[7]

A. Lösungansatz

Mit dem in dieser Arbeit vorgeschlagenen Lösungsansatz wird Entwicklern die Möglichkeit geboten ein bestehendes System um verteilte Methodenaufrufe zu erweitern.
Bei einem verteilten Methodenaufruf wird eine Funktion, eine Berechnung oder ähnliches auf ein entferntes System ausgelagert und dort ausgeführt. Das Ergebnis wird anschließend an den Aufrufer zurückgegeben. Dieses Vorgehen bietet die Möglichkeit die Performance einer Anwendung zu steigern, indem ressourcenintensive Methoden ausgelagert werden. Dabei handelt es sich um Methoden, die sehr viel Hauptspeicher oder CPU-Zeit beanspruchen. Ein weiterer Grund für entfernte Methodenaufrufe sind Sicherheitsanforderungen. Beispielsweise werden alle Berechnungen einer Banking Applikation auf einem gesicherten Server einer Bank durchgeführt.
Durch die Verwendung von verteilten Systemen wird einem System die horizontale Skalierbarkeit hinzugefügt, also die Möglichkeit, Ressourcen von mehreren Rechnern zu verwenden. Durch die Verteilung von Methodenaufrufen erfolgt diese Skalierbarkeit auf Methodenebene.
Dabei sollen - im Gegensatz zum Standardvorgehen - einfache Schnittstellen und die Plattformunabhängigkeit es den Entwicklern ermöglichen mit relativ wenig Aufwand ein bestehendes System zu erweitern.

B. Herausforderungen eines verteilten Systems

Bei der Erstellung eines verteilten Systems müssen verschiedene Risiken betrachtet werden: Das Netz kann unzuverlässig sein und zu Ausfällen führen, die Latenzzeiten und die Bandbreite verzögern den Programmablauf, nicht abgesicherte Netze können die Anwendung kompromittieren, mehrere Administratoren erhöhen die Möglichkeit einer Fehlkonfiguration und die heterogene Systemumgebung erhöht den administrativen Aufwand und bindet Entwicklungskosten.
Da in dieser Arbeit der Fokus auf das Paradigma der Aspektorientierten Programmierung liegt werden nur die der oben genannten Herausforderungen betrachtet, die durch die Softwarebibliothek Thrift gelöst werden können.

C. Apache Thrift

Apache Thrift ist eine Softwarebibliothek zur Entwicklung von effizienten und skalierbaren Backend-Services. Dabei kommt der Codegenerator thrift-gen zum Einsatz, der eine sprachenneutrale Definition von Strukturen und Services ermöglicht und diese in sprachspezifischen Quellcode transformiert. Durch die Sprachneutralität ist die Kommunikation auch zwischen heterogenen Systemen möglich. Thrift überträgt die Daten im Binärformat, ermöglicht aber auch die Auswahl verschiedener Transportprotokolle und zudem auch eine verschlüsselte Übertragung der Daten und Serviceaufrufe. Eine zugesicherte Übertragung von Nachrichten bietet Thrift nicht an.(Siehe [5])
Der Vorteil von Thrift gegenüber dem Java-built-in RPC-Framework, Remote Method Invocation (RMI) genannt, ist die Sprachneutralität sowie die Geschwindigkeit (siehe [6] für genauere Benchmarkergebnisse zu Thrift, Java RMI und andere Serialisierer).

D. Interface Definition Language

Mit der Interface Definition Language (IDL) von Thrift können Datenstrukturen und Schnittstellen, auch Service genannt, deklariert werden. Das Codefragment 1 beschreibt ein Service namens Calculation mit der Methode calculate. Die Methode calculate nimmt als Argument eine Liste von Fließkommazahlen entgegen und gibt als Rückgabetyp eine Fließkommazahl zurück. Aus dieser IDL generiert das Programm thrift-gen spachabhängige Klassen, die im Zusammenhang mit der Thrift-API genutzt werden können.

```
service Calculation {
    double calculate(1: list<double> arg);
}
```

Codefragment 1. Calculation-Service in Thrift IDL

IV. Methodenaufrufe

Die Verwendung von entfernten Methodenaufrufen erfordert im Normalfall die Einbindung der Netzwerkfunktionalität in die bestehende Anwendung. Die Abbildung 2 zeigt einen Methodenaufruf wie er klassischerweise implementiert wird. Eine ressourcenintensive Berechnung soll zur Performancesteigerung auf ein entferntes System ausgelagert werden. Die lokale Methode reicht die Bearbeitung an das entfernte System weiter, auf dem dann die Berechnung durchgeführt wird. Die Ergebnisse werden zurückgesendet und von der lokalen Methode an den Aufrufer zurückgegeben.
Da solche oder ähnliche Funktionen üblicherweise nicht nur an einer Stelle des Programms und nicht nur in einem Projekt genutzt werden sollen, ist es erforderlich viele Teile des Codes anzupassen. Solche Änderungen ziehen häufig weitere Aufwände nach sich. Beispielsweise müssen Unit-Tests angepasst werden, um die neue Funktionen zu testen.
Die Implementierung einer solchen Funktionalität erfolgt dann häufig durch die Erstellung einer zentralen Bibliothek für den gemeinsam genutzten Code. Dieser muss aber dennoch an vielen Stellen und damit in viele Klassen eingebunden werden. Im folgenden wird eine Alternative vorgestellt, welche keine Auswirkung auf den bestehenden Code hat, die aber dem Entwickler die Freiheit gibt, von Methode zu Methode zu entscheiden, ob diese lokal oder auf einem entfernten System ausgeführt werden soll.

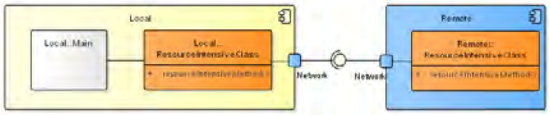


Abbildung 2. Klassischer verteilter Methodenaufruf

A. Separierung der Netzwerkfunktionalität mit AOP und AspectJ

Wie bereits in Abschnitt II erwähnt, können Querschnittsfunktionen mittels aspektorientierter Programmierung von dem restlichen objektorientierten Code separiert werden. Die Netzwerkfunktionalität ist eine solche Querschnittsfunktion, wenn sie an mehreren Stellen im objektorientierten Code eingesetzt wird.
In der hier beschriebenen Lösung wird eine Anwendung lokal programmiert bzw. eine bestehende lokale Anwendung verwendet und angepasst. Die Methoden, welche auf entfernten Systemen ausgeführt werden sollen, werden mit Annotationen von dem Entwickler markiert. Zusätzlich werden diese Methoden mit Pointcuts identifiziert. Ein around Advice ersetzt den lokalen Code durch einen entfernten Methodenaufruf. In diesem Advice wird ein Methodenaufruf über Thrift durchgeführt, welcher an einen Server gesendet wird auf dem die ursprüngliche Funktion, oder eine optimierte Variante dieser Funktion, ausgeführt wird. Innerhalb des Advice wird anschließend das Ergebnis entgegen genommen und dann an den Aufrufer der anfänglich durch den Pointcut identifizierten Methode zurückgegeben. Dieses Vorgehen wird in Abbildung 3 dargestellt.

B. Verwendete Pointcuts

In Abschnitt II-A2 wurde beschrieben wie Pointcuts verwendet werden um Join Points zu identifizieren, an denen ein Advice ausgeführt werden kann. In diesem Beispiel wird eine Methode identifiziert, die durch einen entfernten Methodenaufruf ersetzt werden soll. Der Code im Codefragment 2 zeigt den Pointcut, wie er in dem Beispiel in Abbildung 3 verwendet wird.
Die Syntax der Pointcuts wird in [3] ausführlich beschrieben. Dieser Pointcut identifiziert den Joinpoint: Ausführung (execution) einer bestimmten Methode mit nicht definierten Übergabeparametern (ressourceIntensiveMethod(..) kurz RIM) an welche die Annotation RemoteCall angehängt ist.

```
pointcut remoteCall_RIM(RemoteCall anno,
    List<Double> arg) :
  execution(double
  ressourceIntensiveMethod(..))
  && @annotation(anno) && args(arg);
```

Codefragment 2. Implementierung des pointcut

C. Benutzerdefinierte Annotation RemoteCall

Im vorherigen Abschnitt wurde die Annotation Remote Call erwähnt, welche unter anderem zur Identifikation der Methode im Pointcut verwendet wird. Zusätzlich enthält diese Annotation noch Informationen, die für entfernte Methodenaufrufe wichtig sind, wie Servername und Serverport (siehe Codefragment 3). Durch die Angabe dieser Informationen in den Annotationen jeder Methode ist es außerdem möglich, pro Methode zu entscheiden an welchen Server der Aufruf gesendet werden soll.

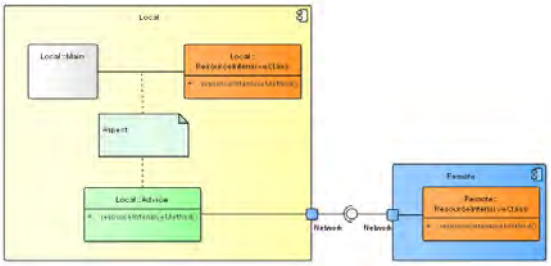


Abbildung 3. Verteilter Methodenaufruf über AOP

```
@Retention(RetentionPolicy.RUNTIME)
@Repeatable(value = RemoteCalls.class)
public @interface RemoteCall {
  String servername();
  int serverport();
  boolean secureTransport() default false;
}
```

Codefragment 3. Implementierung der Annotation

Im Codefragment 3 ist zu sehen dass die Annotation selbst weitere Annotationen enthält. @Retention(RetentionPolicy.RUNTIME) legt fest, dass die Information der annotierten Methode zur Laufzeit zur Verfügung stehen, damit im Advice auf die Informationen zugegriffen werden kann. @Repeatable legt fest das die Annotation mehrfach an eine Methode angehängt werden kann. Die value verweist auf eine weitere Annotations (RemoteCalls) Klasse, welche automatisch verwendet wird wenn die Annotation mehr als ein mal angehängt wird. Diese Klasse enthält als einzigen Parameter ein Array von RemoteCall-Annotationen.
Der Parameter secureTransport kann verwendet werden um im Aspekt Code eine sichere Übertragung zu aktivieren. Dies soll an dieser Stelle lediglich die Möglichkeiten und die Flexbilität aufzeigen, welche sich durch die Verwendung von benutzerdefinierten Annotationen in Verbindung mit Aspektorientierter Programmierung ergeben.

D. Erstellung des Thrift Codes

Damit die Methode auf einem Server mit Thrift aufgerufen werden kann, muss noch der Thrift Code erstellt werden. Der oben erwähnte Codegenerator von Thrift übernimmt die Codegenerierung des für die Übertragung notwendigen Codes. Der clientseitige Code wird im nächsten Abschnitt in das Advice eingebunden. Auf der Serverseite fehlt lediglich die Implementierung der Methode. Da diese in dem Lokalen Code bereits existiert, kann der Code übernommen werden, vorausgesetzt die Serverseite soll ebenfalls mit Java arbeiten. Durch diesen Codegenerator verringert sich der Aufwand für die Erstellung des Client/Server-Code.

E. Around Advice zur Realisierung des Entfernten Methodenaufrufs

Wie bereits erwähnt kann mit dem Around Advice eine Methode ersetzt werden. In dem Beispiel im Codefragment 4 wird der Pointcut remoteCall_RIM aus dem Codefragment 2 verwendet, um darauf basierend ein Advice zu implementieren. Hier werden die Informationen aus der Annotation ausgewertet und an die Thrift-Klasse übergeben. Anschließend wird die generierte Methode des Thrift-Clients aufgerufen, in welcher der entfernte Methodenaufruf realisiert ist. Nach Erhalt des Rückgabewertes wird dieser an den Aufrufer

zurückgegeben. Damit ist diese Implementation der Netzwerkfunktionalität bzw. des entfernten Methodenaufrufs für den Aufrufer vollständig transparent.

```
double around(RemoteCall anno,
    List<Double> arg) :
    remoteCall_RIM(anno,arg) {
  double result = 0;
  Properties properties = new Properties();
  properties.setProperty("host",
    anno.servername());
  properties.setProperty("port",
    anno.serverport());
  Iface clientImpl =
    new ClientImplementation(properties);
  result = clientImpl.calculate(arg);
  return result;
}
```

Codefragment 4. Implementierung des around-Advices

F. Erweiterbarkeit des Lösungsansatzes

Um aufzuzeigen welche Flexibilität dieser Ansatz bietet, wurde die im Codefragment 3 angesprochene Annotation als @Repeatable markiert. Daher kann diese Annotation mehrfach einer Methode hinzugefügt werden. In der Testimplementierung wurde dafür ein zusätzlicher Pointcut und ein zusätzliches Advice implementiert welche das Vorhandensein der in Abschnitt IV-C angesprochenen RemoteCalls Annotation vorausgesetzt. In dem Advice wurde eine Loadbalancing Funktionalität integriert. Somit kann der Entwickler auf sehr einfache Art und Weise ein Loadbalanding auf Methodenebene implementieren. Der Einsatz der mehrfachen Annotation wird im Codefragment 5 dargestellt.

```
@RemoteCall(servername = "10.10.10.11",
    serverport = 9090)
@RemoteCall(servername = "10.10.10.12",
    serverport = 9090)
@RemoteCall(servername = "10.10.10.13",
    serverport = 9090)
public static double
    ressourceIntensiveMethod(){
    return someDouble;
}
```

Codefragment 5. Anwendung der Annotation an einer Methode

V. Future Work

Im vorhergehenden Abschnitt wurde das Konzept dieser Arbeit aufgezeigt und an einem Fallbeispiel umgesetzt. Die Umsetzung benötigt für jede Client-Methode einen Advice und jeder Advice eine eigene Anbindung an einen Server über einen Thrift-Client, welcher die Service-Methode des Servers ausführt. Auch die mit Annotation übergegeben Eigenschaften wie in Abschnitt IV-C müssen dabei ausgewertet und für die Anbindung genutzt werden. Die Kernfunktion des Servers ist die Client-Methode und wird daher 1:1 dupliziert und mit der Server-Implementation von Thrift ergänzt. Insgesamt wiederholen sich viele Codefragmente in unterschiedlichen Ausprägungen. Diese Überschneidungen führen dazu, dass nur wenig Methodenspezifischer Code erforderlich ist, was sich wiederrum für die Generierung von Codefragmenten eignet. Die Trennung von Kernfunktionen und Querschnittsfunktionen mit Hilfe der aspektorientierten Programmierung vereinfacht die Generierung dieser Codefragmente, da clientseitig nur die Aspekte generiert werden müssen. Die Aspekte können dann für verschiedene bestehende Projekte generiert werden.
Daraus ergeben sich zusätzlich zu dem Thrift-Code zwei weitere Komponenten, die generiert werden können:

- der Aspekt-Code
- der Serverseitige-Code

Im folgenden werden die einzelnen Code-Generierungskomponenten erläutert - Abbildung 4 zeigt den Ablauf einer möglichen Codegenerierung auf.

Projekt laden (manuell durchgeführt)
Im ersten Schritt muss das bestehende Projekt geladen werden
Annotation einbinden (manuell durchgeführt)
Im zweiten Schritt wird eine Annotation-Bibliothek eingebunden. Diese Bibliothek enthält die in Abschnitt IV-C beschriebenen Annotationen. Der Entwickler verwendet diese Annotation um jene Methoden zu kennzeichnen, welche mit diesem Lösungsansatz auf entfernte Systeme auszuführen sind.
Methoden mit Annotation identifizieren
Die Annotationen werden mit der Annotation-Processing-API von Java identifiziert und die Methodeninformationen persistiert. Die Methodeninformationen enthalten die Methodensignatur und alle Anweisungen. In diesem Schritt werden die Methoden identifiziert, welche von dem Entwickler mit der RemoteCall Annotation im zweiten Schritt versehen wurden.
Thrift IDL generieren
Mit Hilfe der Methodensignatur wird eine IDL-Datei generiert, die einen Service und die identifizierten Methoden enthält.
***thrift-gen* ausführen**
Die im vorhergehenden Schritt definierte IDL-Datei wird genutzt, um mit dem Programm thrift-gen die Grundlage für das verteilte System zu generieren. In diesen Schritt wird der Client- und Servercode, wie in Abschnitt III-C beschrieben generiert.
Server generieren
In diesem Schritt wird der in dem letzten Schritt generierte Thrift Servercode verwendet. Die Implementation des Thrift-Interfaces wird um die zuvor abgespeicherten Methodenanweisungen ergänzt. Das Server-Projekt wird im Zielordner abgespeichert.
Aspect generieren
Mit Hilfe des Namens der Annotation, der Methodensignatur und dem Clientseitigen Thrift-Code wird der Aspekt generiert. Dabei werden die Codefragmente generiert, die in Abschnitt IV-B und IV-E beschrieben wurden.
Zusammenfügen
Im letzten Schritt wird das bestehende Projekt und der Aspekt durch dem AspectJ-Compiler verwoben und in einem Zielordner abgespeichert.

A. Eclipse Plug-In

Eclipse ist eine integrierte Entwicklungsumgebung (IDE) für zahlreiche Programmiersprachen wie Java und C++ sowie verschiedene Funktionaliäten wie die Model Driven Development Integration und die AspectJ Development Tools. Intern besteht die IDE aus Kernkomponenten sowie Plug-Ins. Für die

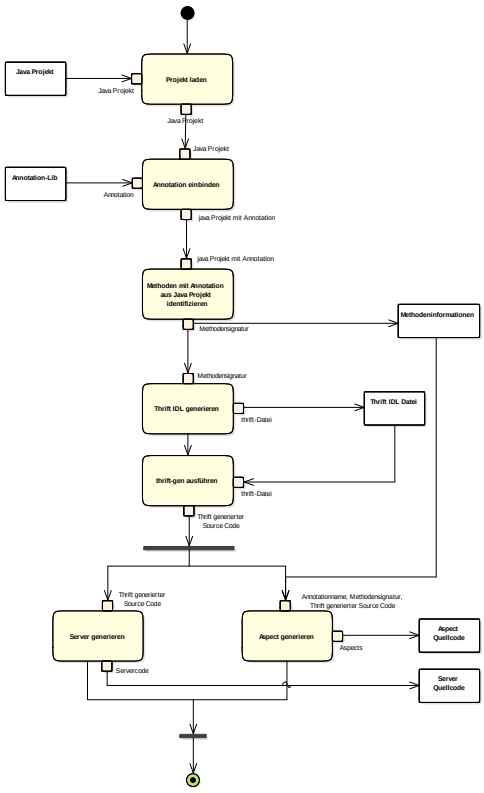


Abbildung 4. Ablauf eines möglichen Codegenerators

Unterstützung des Generierungsprozesses könnte ein Eclipse-Plug-In eine Möglichkeit bieten das beschriebene Toolset in ein bestehenden Entwicklungsprozess einzubinden.

B. Annotation

Die Konfiguration des verteilten Systems kann, wie in Abschnitt IV-F beschrieben, über Annotationen erfolgen und im AspectJ umgesetzt werden.

VI. Related Work

Die Arbeit "Remote pointcut: a language construct for distributed AOP" betrachtet die Ausführung von Aspekten auf entfernten Systemen. Im Gegensatz dazu wird in dieser Arbeit die Ausführung von Methoden auf entfernten Systemen betrachtet.[4]
Die Arbeit "Improving Software Design Using Aspects in Distributed Computing" beschreibt Merkmale der AOP und Merkmale von verteilten Systemen. Die Arbeit ist allerdings sehr abstrakt gehalten und gibt keine konkreten Beispiele.[1] Es wird kein Zusammenhang zwischen AOP und Networking betrachtet und als Framework wird RMI verwendet. In Gegensatz dazu wurde in dieser Arbeit ein funktionsfähiges Beispiel umgesetzt und Apache Thrift als Framework verwendet. Weiterhin wurden in den Pointcuts benutzerdefinierte Annotationen zur Identifizierung der Methoden verwendet.

VII. Fazit

Durch die Kombination von aspektorientierter Programmierung, benutzerdefinierten Annotationen und Thrift für den verteilten Methodenaufruf konnte in dieser Arbeit aufgezeigt werden, wie gut sich bestehende objektorientierte Programme erweitern lassen.
In Abschnitt V wurden dazu zwei Beispiele aufgeführt, die verdeutlichen wie Flexibel der Entwickler durch diese Implementierung der entfernten Methodenaufrufe ist. Es wurde einerseits ein Parameter zu der Annotation hinzugefügt, mit dem sich eine sichere Übertragung ein- und ausschalten lässt. Andererseits wurde ein Load-Balancing implementiert das der Entwickler durch Hinzufügen und Entfernen von Annotationen implementieren und anpassen kann.
Durch die Fokussierung dieser Arbeit auf die Aspektorientierung konnten wichtige Probleme aus dem Bereich Verteilte Systeme nicht adressiert werden. Dieser Lösungsansatz geht von einem immer verfügbaren und störungsfreien Netzwerk aus. Dies entspricht nicht den realen Bedingungen, daher müssen diese Problemstellungen noch näher betrachtet werden. Weiterhin betrachtet dieser Lösungsansatz derzeit ausschließlich den einfachsten Fall einer Methodenverteilung: Das Verteilen einer Methode ohne Abhängigkeiten, also eine statische Methode, welche auf Basis der Übergabewerte das Ergebnis berechnen und zurückgeben kann ohne dabei andere statische Methoden oder Variablen zu verwenden. Auch hier gibt es noch Potential zur Optimierung und Erweiterung des vorgestellten Lösungsansatzes.
Abschließend betrachtet konnte in dieser Arbeit festgestellt werden, dass die aspektorientierte Programmierung viele Vorteile bringt. Dies konnte im Bereich der verteilten Systeme sehr gut eingesetzt werden. Wie oben beschrieben gibt es noch viele Optimierungsmöglichkeiten und Erweiterungen, welche in das hier vorgestellte Konzept eingearbeitet werden können.

Literatur

[1] Deepak Dahiya und Usha Batra. "Improving Software Design Using Aspects in Distributed Computing". In: *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*. IEEE. 2009, S. 882–887.

[2] Gregor Kiczales u. a. *Aspect-oriented programming. Published in proceedings of the European Conference on Object-Oriented Programming (ECOOP). Finland*. 1997.

[3] Ramnivas Laddad. *AspectJ in action: practical aspect-oriented programming*. Greenwich, Conn.: Manning, 2003, XXX, 481 S. ISBN: 1-930110-93-6.

[4] Muga Nishizawa, Shigeru Chiba und Michiaki Tatsubori. "Remote pointcut: a language construct for distributed AOP". In: *Proceedings of the 3rd international conference on Aspect-oriented software development*. ACM. 2004, S. 7–15.

[5] Mark Slee, Aditya Agarwal und Marc Kwiatkowski. "Thrift: Scalable cross-language services implementation". In: *Facebook White Paper* 5.8 (2007).

[6] Eishay Smith u. a. *Comparing varius aspects of Serialization libraries on the JVM platform*. URL: https://github.com/eishay/jvm-serializers/wiki.

[7] Andrew S. Tanenbaum und Maarten van Steen. *Distributed systems: principles and paradigms*. 2. ed. Upper Saddle River, NJ: Pearson, Prentice Hall, 2007, XVIII, 686 S. ISBN: 0-13-613553-6. URL: http://digitool.hbz-nrw.de:1801/webclient/DeliveryManager?pid=3310692.

[8] *The AspectJtm Development Environment Guide*. URL: https://eclipse.org/aspectj/doc/next/devguide/.

Event Processing und Verteilte Systeme

Event Processing Networks

Complex Event Processing mit Hilfe von Event Processing Networks

Martin Haschke, Stefan Retzlaff
Informatik Master
Hochschule Furtwangen
78120 Furtwangen
{Martin.Haschke, Stefan.Retzlaff} @hs-furtwangen.de

Abstract—*Diese Ausarbeitung beschäftigt sich mit dem Thema der Event Processing Networks. Im folgenden wird der Begriff der Event Processing Networks definiert und die verwendeten Komponenten und deren Eigenschaften beschrieben. Darüber hinaus wird das Thema der Skalierbarkeit und den resultierenden Vorteilen aus der Verwendung von Event Processing Networks behandelt. Abgerundet wird das ganze über einen Entwurf für eine grafische Komponentenbeschreibung, da es hierfür noch keinen definierten Standard gibt.*

Keywords—Event Processing; Event Processing Networks; Complex Event Processing; EPN; CEP

I. Einleitung

Unter dem Begriff des Event Processing Networks (EPN) handelt es sich um eine Technik der Analyse und grafischen Darstellung von Complex Event Processing (CEP) unter Verwendung von standardisierten Agents, wodurch die Entwicklung, Verwaltung und Administration einer CEP-Instanz erleichtert werden soll. Hierbei werden die Funktionen des CEP in sehr kleine Teilschritte zerlegt, welche durch Event Processing Agents (EPAs) umgesetzt werden. Die Aufgaben der EPAs orientieren sich anhand eines vorgegebenen Schemas, um die Chance auf Wiederverwendbarkeit möglichst hoch zu halten. Im Detail wird hierauf in Kapitel II,V und VI eingegangen. In Kapitel IV wird anschließend im Detail auf die grafische Darstellung eines EPNs eingegangen. In Kapitel V wird der Begriff des Nested Event Processing Network (NEPN) erklärt. Da es bisher keinen UML Standard oder Ähnliches für die Darstellung von EPNs gibt, wurde sich in dieser Arbeit an den Quellen *Event Processing in Action* [1] und *The power of events* [5] orientiert. Im Zuge der Erstellung dieser Arbeit wurde eine Schablone für Microsoft Visio erstellt, welche dem Design in [1] folgt. Die Grafiken in dieser Arbeit wurden unter Verwendung dieser Schablone erstellt. Die Schablone wird gerne auf Anfrage zur Verfügung gestellt.

Bei dem Begriff Networks werden jedoch nicht nur verteilte Systeme für CEP beschrieben. Ein EPN kann auch Lokal zum Einsatz kommen. Hierbei unterscheidet sich nur die Implementierung der Kommunikation, welche die Agents verwenden. An der Struktur des EPN wird jedoch nichts verändert.

II. Komponenten eines Event Processing Networks

In EPNs kommen Event Producer (EP) und Event Consumer (EC) als Start- und Endpunkte zum Einsatz. Hierbei kann der Weg von Events durch das EPN als Stream bezeichnet werden.

Event Producer stellen den Eingangspunkt in ein Event Processing Network dar. Sie stellen die Quelle für Events in einem Diagramm dar. Eine Quelle kann mehrere darauf folgende Agents mit Nachrichten versorgen. Hierbei können mehrere Arten der Kommunikation verwendet werden. Im Allgemeinen spricht man hierbei von Event Channels. Diese können in 1:1 Verbindungen auftreten, jedoch auch 1:n oder n:m Verbindungen. Auf die Darstellung eines vollständigen Event Channels wird bei 1:1 Verbindungen verzichtet. Zur Reduzierung der Elemente innerhalb einer EPN Darstellung werden hier lediglich Verbindungskanten verwendet.

Event Consumer stellen den Endpunkt eines EPNs dar. Hier enden die Streams und es folgt keine weiteren Event Channels. Event Consumer reagieren in der Regel auf ankommende Events, indem sie zum Beispiel eine Email versenden oder einen Alarm auslösen.

Event Channels werden zur Kommunikation zwischen den Komponenten eines EPNs verwendet. Es können mehrere Optionen für Event Channels gesetzt werden. Hierbei kommen folgende Techniken zum Einsatz: Event Queue, function call, publish-subscripe Topic. Je nach gewählten Optionen kann ein Event Channel mehrere Eingänge und mehrere Ausgänge besitzen. Zudem kann auf den Event Channels durch Routing Schemas gezieltes verarbeiten Anhand folgender Schematypen eingesetzt werden:

- Fixed: jedes Event wird geclont und auf allen Ausgängen ausgegeben
- Type based: Routing Entscheidung basiert auf Event Type
- Content based: Routing Entscheidung basiert auf Inhalt / Werten des Events

Dieses Regelwerk kann auch kombiniert werden, so dass komplexere Routing Schemes eingesetzt werden können. Hierbei spricht man dann von impliziten Channels, welche keinerlei Logik besitzen und expliziten Channels, die simple Logik, wie beispielsweise Routing Logik, beinhalten.

Zur Verarbeitung der Events werden *Event Processing Agents* (EPA) eingesetzt. Jeder Agent steht hierbei für eine

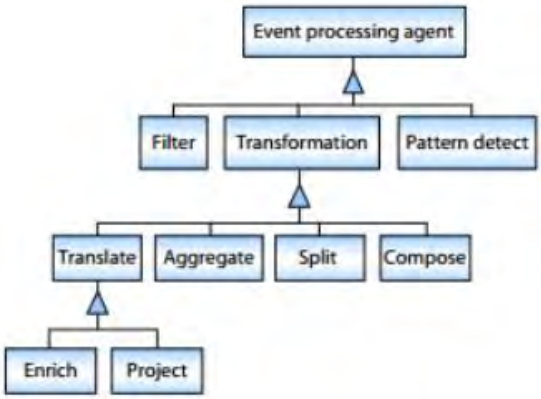


Abbildung 1: Unterschiedliche Typen von EPAs [1, S.52]

kleine Teilaufgabe, welche sich anhand eines festen Schemas orientiert. Die ersten und einfachsten EPAs sind die *Filter Agents*, welche Filterfunktionen durchführen. An dieser Stelle wird der Stream auf gewisse Eigenschaften gefiltert. Da kein Event verloren gehen darf, hat dieser Agent in der Regel ein input Terminal und drei output Terminals. Hierbei wird lediglich zwischen „Filter trifft zu“ und „Filter trifft nicht zu“ unterschieden. Wie in Abbildung 3 zu sehen ist, dient das dritte output Terminal der Aussortierung von nicht filterbaren Events. Der Inhalt der Events wird nicht verändert. Hier kommt noch ein weiteres Detail zum tragen. In der Darstellung der Komponenten kommen noch kleine Pfeile an den Enden einer Event Channel Verbindung zum Einsatz, um die input und output Streams von Elementen eindeutig zu Kennzeichnen. Diese werden im folgenden Terminals genannt. In deren Darstellung unterscheiden sich auch die Quellen [1] und [5].

Die nächste Art von Agents werden *Transformation Agents* genannt. Sie verändern den Inhalt der verarbeiteten Events. Da es einige Möglichkeiten gibt, Events zu verändern, gibt es hier noch weitere Unterscheidungen, welche in Abbildung 1 dargestellt sind. Unter *Translate Agents* werden zwei weitere Agents genannt: *Enrich* und *Project*. Translate Agents sind Agents, welche die Sprache des Inhalts eines Events verändern. Hierbei können sie Informationen zu einem Event hinzufügen, was dann ein Enrich Agent wäre. Diese Informationen werden in der Regel statisch hinzugefügt oder aus Global State Elements abgerufen, auf welche später genauer eingegangen wird. Im Gegenzug gibt es Project Agents, welche die Menge an Informationen innerhalb des Elements durch löschen von Informationen reduzieren. Diese beiden Agents sind Stateless Agents, auch zustandslose Agents genannt. Im Gegensatz hierzu gibt es Stateful Agents, auch zustandsbehaftete Agents genannt. Diese Agents arbeiten mit Informationen, welche nicht auf ein einzelnes Event beschränkt sind. Hierzu zählt die einfache Version eines Zählers oder auch die Durchschnittsberechnung über ein bestimmtes Attribut von mehreren Events. Ein weiterer wichtiger, zustandsbehafteter Agent ist der *Pattern Detect Agent*, welcher Complex Event Processing über mehrere Events betreibt.

Die nächste größere Gruppe an Agents sind die *Aggregation Agents*. Aggregation Agents bilden eine Funktion über mehrere Events ab. Bekannte Beispiele sind hier Summenbildung, Durchschnittswertbildung, Minimum und Maximum über ein definiertes Fenster aus einer bestimmten Anzahl an Events oder einem bestimmten zeitlichen Rahmen, generiert aus dem Inhalten mehrerer Events. Als nächstes werden die *Split Agents* betrachtet. Diese Agents teilen Events in mehrere Events auf, indem sie die Informationen auf mehrere Events nach einem bestimmten Muster aufteilen. Oder sie erstellen Klone von Events, damit zwei Agents mit demselben Event arbeiten können. Die nächsten Agents, welche bearbeitet werden, sind die *Compose Agents*. Diese Agents sammeln mehrere Events, kombinieren diese und geben die Informationen in einem Event weiter, ohne den Inhalt der Informationen zu verändern. Die letzten Agents, welche betrachtet werden, sind die Pattern Detect Agents. Ein Pattern Detect Agent sucht nach Mustern über mehrere Events hinweg. Seine Funktion / Arbeitsweise kann man über ein Nested EPN erläutern. Hierbei kann komplexe Logik zum Einsatz kommen, um sozusagen logische Schlussfolgerungen und einen Informationsgewinn zu erzeugen.

Unter dem Begriff *Event Processing Terminals* werden die Schnittstellen zwischen den einzelnen Komponenten und den Event Channels bezeichnet. Diese werden in Form kleiner Dreiecke symbolisiert - ihre Richtung gibt den Hinweis, ob es sich um eine input oder output Schnittstelle handelt. In der Regel besitzt jeder Agent mindestens ein Input Terminal und mehrere Output Terminal.

Global State Elements stellen über Agenten hinweg Informationen für mehrere Agents bereit. Hierbei kommen verschiedene Formen vor. Die einfachste Form ist die Reference Data Form. Hierbei wird einfach statisch eine Information für mehrere Agents bereit gestellt. Eine weitere Form stellt der Event Store dar. An dieses Global State Element können Events geschickt werden, welche anschließend auch wieder abgerufen werden können. Hierbei werden alle empfangenen Events mit einem zeitlich geordneten Speicher gesichert und können jederzeit wieder abgerufen werden. Eine einfachere Form stellt der Event Processing State dar, welcher dieselben Funktionen wie der Event Store anbietet. Allerdings wird hier nur jeweils ein Event bevorratet. Die letzte Form stellt der State of external Entities dar. Hier werden Informationen aus Event Processing Network fremden Quellen bereit gestellt, wie zum Beispiel das aktuelle Wetter, ein Wert einer Aktie an der Börse, etc.

III. Beispiel Fleet Control

Die Funktionsweise eines Event Processing Networks und seiner Komponenten wird in dieser Arbeit an dem Beispiel eines Fleet Control Systems veranschaulicht. Dieses wird mit Hilfe eines EPN Diagramms in Abbildung 2 dargestellt. Es zeigt ein Event Processing System, welches die Fahrten von Trucks überwachen soll. Als Reaktion soll ein Kontakt mit einer Nachricht (Target Mailer) informiert werden, kurz bevor der Truck an seinem Ziel eintrifft. Weiterhin soll die Zentrale (Fleet Control) informiert werden, wenn es einen begründeten Verdacht auf eine Verspätung des Trucks, beziehungsweise Probleme mit dem Truck, gibt. Zu diesem Zweck produziert der Truck GPS Daten (Truck GPS) und Status Informationen (Truck), welche über das Fahrzeug an das EP System geleitet werden. Zusätzlich werden noch Verkehrsinformationen (Traffic Information) verarbeitet. Aus den stetig eingehenden GPS Informationen wird eine Durchschnittsgeschwindigkeit berechnet. Anhand dieser und der ursprünglichen GPS Information des Trucks wird ermittelt, wann der Truck bei seinem

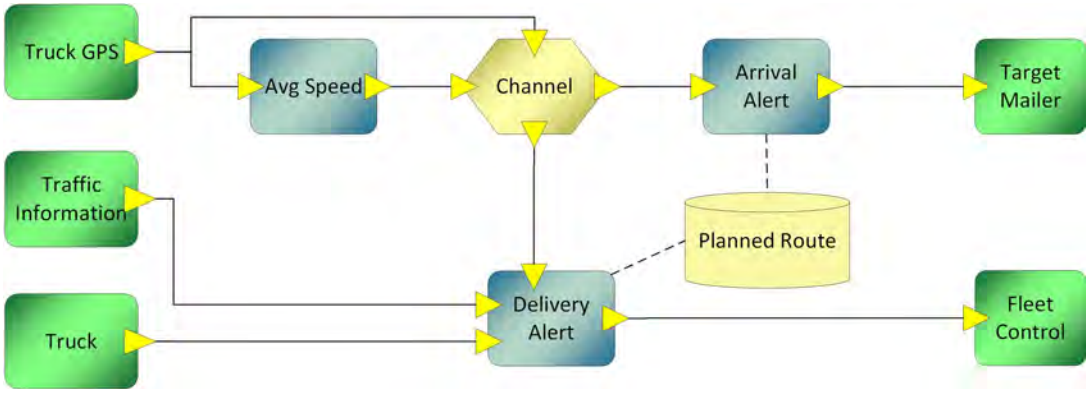


Abbildung 2: Beispiel EPN - Fleet Control

Fahrtziel voraussichtlich ankommen wird. Die gesammelten Informationen über die Durchschnittsgeschwindigkeit, die Verkehrsinformationen und der Status des Trucks werden verwendet, um vorherzusagen, ob der Truck eine Verspätung haben könnte oder ob dieser aktuell Probleme hat. Daraufhin könnte die Zentrale benachrichtigt werden

IV. Darstellung der Komponenten im Event Processing Network

Alle bisher kennengelernte Elemente eines Event Processing Networks können grafisch dargestellt werden. Diese grafische Darstellung wird verwendet um die Übersichtlichkeit und Verständlichkeit zu steigern, da eine Grafik in der Regel einfacher und schneller zu verstehen ist als die dargestellten Informationen textuell zu vermitteln. Eine textuelle Darstellung kann ebenfalls gewählt werden, jedoch wird diese sehr schnell unübersichtlich und eine Abstraktion der Informationen, welche zum Verständnis des EPNs nötig sind, wird erschwert.

In Abbildung 2 werden alle Elemente einer grafischen Darstellung eines EPNs gezeigt: Event Producer, Event Consumer, Event Processing Agents, Message Channel und Global State Elements. Auf der linken Seite, zu Beginn des Streams, sind die Event Producer zu sehen. Und auf der rechten Seite, gegen Ende des Streams, befinden sich die Event Consumer. Mit den Bezeichnungen „Avg Speed“, „Delivery Alert“ und „Arrival Alert“ sind die Event Processing Agents in Abbildung 2 zu finden. Zudem ist ein Event Channel mit der Bezeichnung „Channel“ zu finden. Mit der Bezeichnung „Planned Route“ ist ein Global State Element zu finden.

Eine detailliertere Beschreibung der Funktion eines EPAs ist in Abbildung 3 und 4 zu sehen. Abbildung 3 zeigt einen Filter EPA und dessen internen Aufbau. Hier ist deutlich zu sehen, dass der EPA ein input Terminal besitzt und drei output Terminals. In Abbildung 4 ist die detaillierte Darstellung eines Transformation EPAs abgebildet. Hier gibt es ebenfalls nur ein input Terminal, jedoch nur zwei output Terminals. Die ausgeführten Funktionen auf die Events können mit Hilfe von Kästchen dargestellt werden, damit das Zusammenspiel innerhalb des Agents übersichtlich bleibt. Diese Darstellung hilft sehr bei der Visualisierung von Pattern Detect Agents, da diese in der Regel komplexes Event Processing betreiben, welches aus mehreren miteinander interagierenden Modulen besteht. Aus diesem Grund werden für Pattern Detect Agents gerne Nested Event Processing Networks verwendet.

V. Nested Event Processing Networks

Wie bereits erwähnt, können die Event Processing Agents viele verschiedene Aufgaben haben. Idealerweise sollte die Funktion eines EPAs immer nur auf eine Aufgabe beschränkt sein. Bei komplexeren Aufgabenstellungen, welche mehrere Teilaufgaben beinhaltet, würden diese Aufgaben auf mehrere Agents verteilt werden. Stellt man all diese Agents in einem großen vollständigen Diagramm dar, würde dieses sehr komplex und unübersichtlich werden. Daher gibt es die Möglichkeit der Nested Event Processing Networks.

Dabei wird ein EPN, welches einen Teil eines anderen EPNs darstellt, in diesem durch einen Event Processing Agent repräsentiert. Die input und output Terminals des Agents werden in dem Netzwerk entsprechend durch Producer und Consumer dargestellt. Dazwischen können wiederum beliebig viele Agents eingesetzt werden. Weiterhin besteht die Möglichkeit, dass jeder dieser Agents ein weiteres EPN darstellt. Dies ermöglicht dem Designer ein komplexes EPN aufzuteilen und für

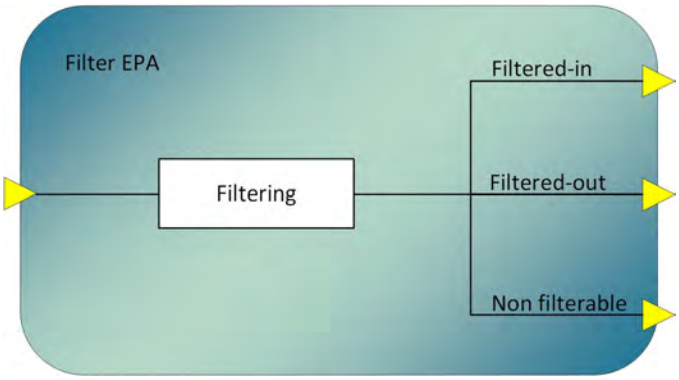


Abbildung 3: Filter EPA

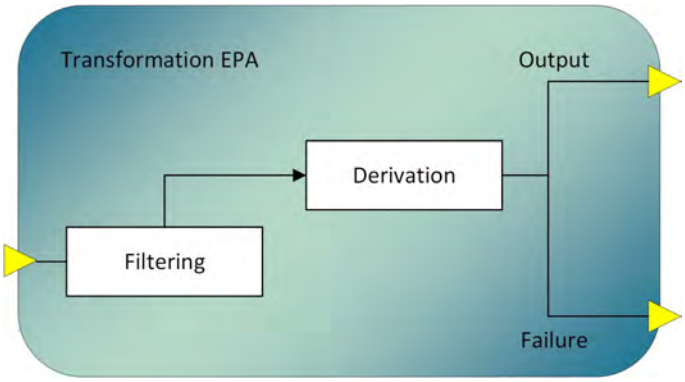


Abbildung 4: Transformation EPA

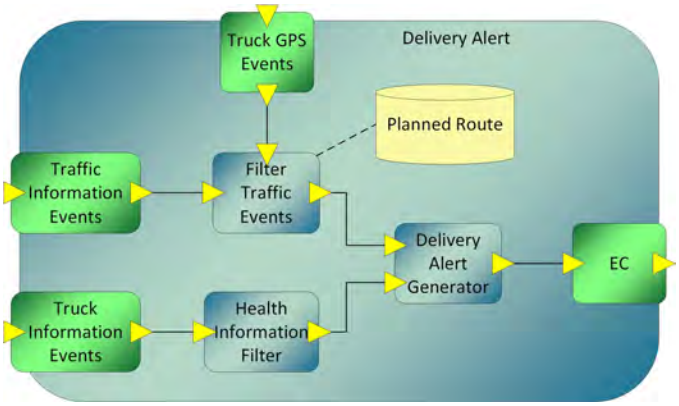


Abbildung 5: Nested Event Processing Network - Delivery Agent

den Betrachter übersichtlich darzustellen. Genauer gesagt wird dieser hierarchische Ansatz verwendet, um die Funktionen der Agents bis hin zur feinsten Granularität darzustellen, ohne die für eine graphische Darstellung sehr wichtige Übersichtlichkeit zu verlieren.

Wird der Event Processing Agent „Delivery Alert“ aus dem Beispiel genauer durchleuchtet, sind folgende Komponenten zu finden: Dieser Agent verfügt über drei Input Terminals und ein Output Terminal. In Abbildung 5 wird ein untergeordnetes EPN dargestellt, welches die Funktionsweise dieses Agens verdeutlicht. In dem untergeordneten Diagramm, wie in Abbildung 5 dargestellt, ist zu sehen, dass die Terminals durch Producer und Consumer dargestellt werden. Es fällt auf, dass der Producer „Truck GPS Events“ des untergeordneten EPNs in den übergeordneten EPN mit einem Channel verbunden ist. Nun könnte der Producer auch durch einen Channel ersetzt werden. Allerdings ist es in diesem Beispiel sinnvoller, einen Producer zu wählen, welcher verdeutlicht, welche Events des Channels hier tatsächlich verwendet werden. Weiterhin werden in diesem einfachen Beispiel die Traffic Information Events gefiltert, da nur jene interessant sind, welche sich auf der Route des Trucks noch ereignen werden, die dieser noch vor sich hat. Der „Delivery Alert Generator“ verwendet die Verkehrsinformationen und die Statusinformationen des Trucks, um komplexe Events zu generieren. Diese werden erzeugt wenn erkannt wurde, dass sich die Auslieferung verzögern könnte oder Probleme mit dem Truck erkannt wurden.

VI. Skalierungsansätze

Zur Laufzeit können die Komponenten des Event Processing Networks mehrere Ausprägungen haben. Dies wird schon an dem Beispiel deutlich. Schließlich ist davon auszugehen, dass es mehrere Trucks gibt, welche gleichzeitig unterwegs sind und überwacht werden sollen. Es wäre allerdings sehr unübersichtlich, würde man versuchen, dies in einem EPN Diagramm zu visualisieren. Stattdessen wird davon ausgegangen, dass die Komponenten zur Laufzeit beliebig viele Instanzierungen, auch Runtime Artifacts genannt, haben können. Dies schafft einige Herausforderungen, welche in diesem Kapitel näher beleuchtet werden.

Um die einzelnen Runtime Artifacts der EPAs zur Laufzeit möglichst effizient auf verschiedene Systeme verteilen zu können, ist es unerlässlich, frühzeitig bei dem Design der EPAs einige Punkte zu beachten. Die EPAs werden unter Verwendung von Nested EPNs so lange in untergeordnete EPNs aufgeteilt, bis jeder Agent nur noch eine elementare Aufgabe hat, welche sich nicht mehr weiter unterteilen lässt. Diese Aufgabe kann dann idealerweise durch Event Pattern dargestellt und später implementiert werden. Für die Agents auf dieser Abstraktionsebene sind folgende Überlegungen durchzuführen:

A. *Pipelined vs Partitioned Event Processing*

Beim Pipelined Event Processing wird davon ausgegangen, dass eine Aufgabe in mehrere kleinere Unteraufgaben unterteilt werden kann, welche nacheinander abgearbeitet werden können. Im EPN werden zu diesem Zweck mehrere Agents hintereinander geschaltet. [2, S.228]

Wenn Aufgaben gleichzeitig und unabhängig voneinander durchgeführt werden können, wird vom Partitioned Event Processing gesprochen. Dabei werden Aufgaben auf verschiedene Agents aufgeteilt und bearbeitet. Die Ergebnisse werden später wieder zusammengeführt. [2, S.227]

An dieser Stelle ist noch einmal zu erwähnen, dass es dabei um unterschiedliche Aufgaben geht, welche auf unterschiedliche EPAs verteilt werden und nicht um gleiche Aufgaben, welche auf unterschiedliche EPA-Instanzen verteilt werden. Dies ist bei der Implementation zwar auch zu berücksichtigen, wird aber nicht im EPN Diagramm dargestellt.

B. *Stateful vs Stateless Event Processing Agents*

Ein Stateless EPA bearbeitet immer nur ein Event und benötigt keinen Bezug zu vorhergehenden oder nachfolgenden Events[1, S.44f]. In anderen Worten ist es nicht notwendig, Zustandsinformationen zu halten, um diese für die Verarbeitung anderer Events wiederzuverwenden. Durch diese Zustandslosigkeit ist diese Art von Agents sehr einfach Skalierbar, da zur Laufzeit problemlos mehrere Instanzen dieser Agents gestartet werden können, um die zu verarbeitenden Events auf diese zu verteilen. Beispielsweise wird der „Health Information Filter“ aus Abbildung 5 nur die Status Informationen des Trucks weiterleiten, welche eine Verzögerung bewirken können. Auf diese Weise wird der nachfolgende Agent entlastet, da nur noch relevante Informationen bei diesem ankommen.

Wie man auch an diesem Beispiel gut erkennen kann, ist es wichtig, Funktionen, welche von Stateless Agents ausgeführt werden können, durch eigenständige Agents im EPN abzubilden. Dies wird später bei der Implementation der Agents die Skalierung stark vereinfachen.

Im Gegensatz dazu können Stateful Agents mehrere eingehende Events verarbeiten, um diese in einem gemeinsamen Kontext zu betrachten. Beispielsweise wird der „Avg Speed“ Agent mehr als ein GPS Event benötigen, um eine Durchschnittsgeschwindigkeit zu berechnen. Angenommen, dass dieser Agent den Durchschnitt der letzten fünf Minuten errechnen soll, benötigt dieser zusätzlich zu der aktuellen GPS Position die GPS Positionen der letzten fünf Minuten. Der Agent muss also die GPS Events fünf Minuten lang vorhalten um stetig die Berechnung durchführen zu können.

Wird die Skalierbarkeit solcher Agents betrachtet, so wird klar, dass sich diese nicht so einfach auf mehrere Systeme verteilen lassen. Denn es muss sichergestellt werden, das alle Events, welche für einen Stateful Agent notwendig sind und logisch zusammen gehören, bei dem selben Agent ankommen. In dem Beispiel würde dies bedeuten, dass zur Laufzeit die

Events einer „Truck GPS“ Instanz, also die Events des selben Trucks, bei derselben „AvgSpeed“ Agent Instanz ankommen müssen. Eine mögliche Umsetzung wäre die Verteilung auf die verschiedenen Instanzen des „AvgSpeed“ Agents an eine Quell-ID zu binden. Da die Übertragung der Events immer über Channels erfolgt, welche in der Lage sind, Routing Entscheidungen auf Basis des Kontents zu treffen, könnte diese Lösung ohne weitere Komponenten implementiert werden. Diese Problematik kann, je nach Komplexität der Korrelation, teilweise, vollständig oder auch gar nicht mit Hilfe der Implementierungssprache gelöst werden[1, S.136]. Das Design des EPN sollte allerdings Implementationsunabhängig erfolgen.

C. Dynamische Skalierung

Wie bereits erwähnt, hat eine Komponente zur Laufzeit mehrere Runtime Artifacts. Daher kann sich die Anzahl der Instanzen ändern, wenn sich die Menge der eingehenden Events ändert. In [3] wird ein Konzept vorgestellt, das ein Event Processing Network beschreibt, welches sich zur Laufzeit verändert. Es werden einzelne oder mehrere Komponenten zur Laufzeit gestartet und gestoppt. In dieser Arbeit wird hauptsächlich von einem Event Processing System ausgegangen, welches für ein Business Process Management System ausgelegt ist. Grundlegend lässt sich die Idee allerdings auch auf das Beispiel aus Abbildung 2 übertragen.

Angenommen, dass neue Event Producer erzeugt werden („Truck“ und „Truck GPS“), sobald ein Truck seine Tour startet, so wird ab diesem Zeitpunkt die Verarbeitung der Events dieser Quellen notwendig. Es könnten also zu diesem Zeitpunkt neue Instanzen der EPAs gestartet werden um die Events zu verarbeiten. Das würde die Problematik mit der Verteilung der Stateful EPAs entschärfen, da jeder Truck seinen eigenen EPN Baum verwenden würde. Natürlich wäre dies sehr Ressourcen intensiv. Davon ausgehend, dass ein Truck nicht notwendigerweise ein eigenes EPN benötigt, sondern lediglich eines mit ausreichend Ressourcen um die Events verarbeiten zu können, so könnten die neuen Events entweder mit Hilfe der Channels durch ein bestehendes EPN geroutet werden oder es werden neue Instanzen gestartet.

Dieser theoretische Ansatz zur Skalierung des EPNs zur Laufzeit müsste nicht notwendigerweise für das gesamte EPN erfolgen, sondern könnte auch auf einzelne Agents oder NEPNs angewendet werden.

VII. Tools

Auf dem Markt finden sich einige Programme, mit denen Event Processing Networks erstellt werden können. Einige davon sind in den Entwicklungsumgebungen integriert und bieten eine direkte Möglichkeit, Agent Typen zu definieren und Sourcecode zu generieren. In Zuge dieser Arbeit wurde die Software JDeveloper mit Oracle Event Processing (OEP)[4] und das Eclipse Plugin XChart[6,7] betrachtet. Die hier beschriebenen Komponenten finden sich in beiden Tools wieder, allerdings werden die Komponenten aufgrund des fehlenden Standards unterschiedlich benannt. Beispielsweise werden die Agents im OEP als Processors und die Producer und Consumer als Adapter dargestellt. Beide Tools verfügen über Foreward Engineering Möglichkeiten, also über Funktionen zum Exportieren von Quellcode auf Basis des erstellten EPN Diagramms.

VIII. Ausblick

Zusammenfassend ist der Lösungsansatz der Event Processing Networks, um die auftauchenden Probleme mit komplexem Event Processing und dessen Folgen zu lösen, sehr vielversprechend. Das Design eines Systems für komplexes Event Processing wird erleichtert und überschaubar. Zudem werden die Streams und deren dazugehörige Agents transparenter, was den Betrieb, die Wartung und das Vertrauen der Benutzer in das System verbessert. Ebenfalls wird die Skalierung eines Event Processing Systems enorm erleichtert, da jeder Agent für eine einzelne Aufgabe steht und daher lediglich die Anzahl der Agenten und deren Instanzen auf Systeme verteilt werden muss, um das System seinen wachsenden Anforderungen anzupassen. Aufbauend auf diese Arbeit wäre eine Ausarbeitung eines Standards für die Darstellung und das Design von Event Processing Networks wünschenswert, da bei der Recherche mehrere Darstellungsmöglichkeiten gefunden wurden, welche jedoch gemeinsame Komponenten aufweisen. In dieser Arbeit wurde auf die textuelle Darstellung von Event Processing Networks verzichtet, da diese Darstellung lediglich ergänzend und nicht gleichwertig gehandhabt werden sollte.

References

[1] O. Etzion and P. Niblett, *Event processing in action*. Greenwich, Conn.: Manning, 2011.

[2] A. Alves, R. J. Smith, and L. Williams, Getting Started with Oracle Event Processing 11g: Create and develop real-world scenario Oracle CEP applications. Birmingham: Packt Publishing, 2013.

[3] Maximilian Lackaw, “Konzeption und Realisierung eines generischen Event-Mechanismus für Prozess- Management-Systeme,” Diplomarbeit, Institut für Datenbanken und Informationssysteme, Universität Ulm, Ulm, 2012.

[4] Oracle Event Processing. Available: http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/index.html.

[5] D. Luckham, The power of events: An introduction to complex event processing in distributed enterprise systems, 6th ed. Boston, Mass.: Addison-Wesley, 2010.

[6] Carlos Neves Júnior, xchart: A tool for specification of Distributed Event-based Systems (DEBS) and behavioral modeling (Statecharts). Available: https://code.google.com/a/eclipselabs.org/p/xchart/.

[7] Carlos Neves Júnior, Hans Kurt Edmund Liesenberg, “A tool for development of distributed event-based systems,” Instituto de Computação, Universidade Estadual de Campinas, Campinas–SP–Brazil.

Denial of Service Angriffe auf SOAP Web Services

Schwachstellen und Verteidigungsmaßnahmen

Martin Haschke, Stefan Retzlaff
Informatik Master
Hochschule Furtwangen
78120 Furtwangen
(martin.haschke, stefan.retzlaff) @hs-furtwangen.de

Prof Dr. Friedbert Kaspar
Hochschule Furtwangen
78120 Furtwangen
friedbert.kaspar@hs-furtwangen.de

Abstract—Diese Ausarbeitung beschäftigt sich mit Denial of Service Angriffen welche speziell auf SOAP Web Services ausgerichtet sind. Es wird die Verwundbarkeit von den Java basierten Web Service Frameworks CXF und Axis2 gegen diese Angriffe untersucht. Hierbei werden bereits bekannte Schwachstellen auf deren Aktualität validiert. Weiterhin werden mögliche Verteidigungsmaßnahmen vorgestellt und im Bezug auf Umsetzbarkeit, Effektivität und Kostenaufwand analysiert und verglichen.

Keywords — SOA; Software Oriented Architecture; Denial of Service; DoS; WS-Attacker; Security; Web Service; SOAP, CXF, Axis2

I. Einleitung

Der Austausch von Daten ist in der heutigen Zeit essentieller Bestandteil des Internets geworden. Hierbei handelt es sich nicht mehr nur um das simple Abrufen von Informationen, welche auf Web Servern bereit gestellt werden, sondern um interaktiven Datenaustausch, welcher meist zwischen zwei Parteien stattfindet. Gerade im gewerblichen Umfeld ist hiermit eine Wertschöpfung verbunden, wodurch diese Kommunikation selbstständig, standardisiert und kontinuierlich verfügbar sein sollte. Hier setzt diese Arbeit an und beschäftigt sich mit Serviceorientierten Architekturen mit Hilfe von SOAP. Dieses Protokoll bietet Zustandslosigkeit, ist im Rahmen des W3C standardisiert und kann beliebig erweitert werden. Der Ablauf einer solchen SOAP Kommunikation wird in Kapitel II genauer beleuchtet. Zudem kann der eventuell bereits bestehende Web Server um diesen Service erweitert werden. Durch diese Eigenschaften wird ein SOAP Service allerdings auch zum Angriffsziel. An diesem Punkt setzt diese Arbeit an. Es werden Denial of Service Angriffe beleuchtet, welche auf der Ebene von SOAP arbeiten. Diese Angriffe werden ebenfalls in Kapitel II detailliert beschrieben. Auf die Möglichkeit von Denial of Service Angriffen auf tiefer liegenden Ebenen, wie TCP-Syn-Flood, wird bewusst nicht eingegangen.

Da mit dem Ausfall eines solchen Web Services finanzielle Verluste einhergehen, sollte dies vermieden werden. Allerdings ist die Absicherung eines Web Services nicht durch das Einsetzen einer Firewall erledigt. Hierbei muss bei den Komponenten des Web Services angesetzt werden, da klassische, statische Firewalls auf diese Angriffe nicht ansprechen, denn sie arbeiten auf anderen Schichten des ISO/OSI-Modells[9]. Potenzielle Angreifer könnten auf diese Weise ein bereits gehärtetes System zum Ausfall bringen, da auf Komponenten abgezielt wird, welche für die Verarbeitung der Daten auf dem Web Service verantwortlich sind.

Um die Ausmaße solcher bekannter Angriffe messbar zu machen, wurden zwei Web Services mit unterschiedlichen Implementierungen aufgesetzt. Diese wurden mit Hilfe des Werkzeugs WS-Attacker [7] angegriffen. Auf die Testdurchführung und die daraus resultierenden Ergebnisse wird in Kapitel IV eingegangen.

Das Problem bei Denial of Service Angriffen auf Applikationsebene liegt darin, dass klassische Firewall-Systeme auf diese Angriffe nicht reagieren. Die Anfragen, welche der Angreifer an den Service schickt, unterscheiden sich von legitimen Anfragen normaler Benutzer nur sehr gering. Zudem entsprechen sie keinem wiederkehrenden Muster, wodurch ein Angriff zu erkennen wäre. Durch diese Eigenschaft ergibt sich eine ähnliche Problematik wie bei Cross Site Scripting Angriffen. Sie sind für klassische Firewall Systeme nicht zu erkennen und können daher nicht unterbunden werden, da der Angriff sich innerhalb eines definierten Regelwerks bewegt. Eine Lösung mit Web Application Firewalls und Empfehlungen zur Abhärtung von Web Services inklusive einer Zusammenfassung dieser Thematik ist in Kapitel V zu finden.

Dieser Beitrag analysiert, ob die Sicherheitsprobleme, welche die Arbeit [1] in Bezug auf Denial of Service Angriffe aufgedeckt hat, behoben wurden oder weiterhin bestehen.

II. Grundlagen

In diesem Kapitel wird auf die Grundlagen von SOAP basierten Web Services eingegangen. Des weiteren werden die verwendeten Angriffe mit den verbundenen Schwachstellen im Detail erklärt.

A. *Denial of Service Angriffe*

Denial of Service (DoS) Angriffe beschreiben Angriffsmuster, welche darauf abzielen, die Erreichbarkeit eines Services zu beeinträchtigen. Ein Service kann beispielsweise eine Webseite oder ein E-Mail Service sein. Es gibt verschiedene Arten von Denial of Service angriffen. In dieser Arbeit werden jene betrachtet, welche sich gezielt gegen Web Services richten.

B. *SOAP Basierte Web Services*

Es gibt viele Definitionen für Web Services. Das World Wide Web Consortium (W3C) definiert Web Services wie folgt: *"A Web service is a software system designed to support*

interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.[4]"

Einfacher gesagt bieten Web Services eine standardisierte Schnittstelle um auf Funktionen einer Applikation von anderen Services oder Anwendungen aus zugreifen zu können. Es werden dabei verschiedene Übertragungsprotokolle unterstützt, wobei die DoS Angriffe, welche in dieser Arbeit untersucht werden, sich ausschließlich gegen SOAP basierte Web Services richten. Web Services werden üblicherweise über die Web Service Description Language (WSDL) beschrieben. Dabei handelt es sich um eine auf XML basierenden Sprache. Detailliertere Informationen zu XML kann in [5] nachgeschlagen werden.

In einem WSDL Dokument werden unter anderem die Schnittstellen beschrieben, über welche der Web Service aufgerufen werden kann. Da dieses Dokument standardisiert ist und zusammen mit einem Web Service veröffentlicht wird, kann jeder Client der den Web Service kennt, dieses Dokument herunterladen. Auf diesem Wege werden dem Client die verfügbaren Schnittstellen und Funktionen bekannt gegeben. Weiterhin erhält der Client Anweisungen, welche Protokolle und Standards verwendet werden sollen. Zusammengefasst beschreibt das WSDL File die Konditionen des Vertrages zwischen Client und Server.

SOAP ist ein Standard zur Übertragung von Web Service Nachrichten. Es ist auf der Schicht 6-7 des ISO/OSI Modells angesiedelt. Es kann prinzipiell mit beliebigen Protokollen der Sitzungsschicht verwendet werden. Am häufigsten wird hier das HTTP Protokoll verwendet. Aus diesem Grund laufen Web Services häufig auf Web Servern. Auf diese Weise können Web Services einfach in bestehende Strukturen integriert werden.

SOAP Nachrichten bestehen aus einem Umschlag (Envelope) in welchem sich evtl. mehrere Header und der Body der Nachricht befinden. In den Headern stehen Informationen über den Nachrichteninhalt wie Verwendungszweck, Routing, Informationen über Authentifizierung oder eine mögliche Verschlüsselung. In dem Body werden die eigentlichen Nutzdaten übertragen.[5]

Bei einem typischen Ablauf beginnt der Client bei einer SOAP Nachrichtenübertragung mit einer Anfrage an den Server. Hierbei lädt er das WSDL File vom Server, wodurch er die benötigten Informationen zur Verwendung des angegebenen Services erhält. Gemäß der Schnittstellenbeschreibung formuliert der Client dann die SOAP Nachricht für den Web Service. Dieser nimmt die Daten entgegen, liest sie ein, verarbeitet diese und antwortet dem Client. Dabei lassen die Regeln, welche in der WSDL Datei beschrieben sind, Spielraum für Angriffe, um den Web Service und das darunter liegende System außer Betrieb nehmen zu können.

C. Web Service spezifische DoS Angriffe

In diesem Kapitel werden die SOAP Web Service spezifischen Angriffe im Detail erklärt. Der Vollständigkeit halber werden alle DoS Angriffe, welche in WS-Attacker[7] zur Verfügung stehen, behandelt.[1,5,8]

Coercive Parsing Attack

Bei einem Coercive Parsing Angriff wird der Web Service mit einem SOAP Request in Form einer XML-Datei belastet, welche viele verschachtelte XML-Tags beinhaltet. Durch die Verarbeitung einer solch präparierten XML -Datei wird auf dem angegriffenen System eine sehr hohe Last erzeugt. Der Angriff zielt auf den XML Parser des Systems. Besonders Anfällig für Angriffe sind DOM-basierte XML Parser. Kommt ein solcher Parser zum Einsatz kann das Zielsystem stark überlastet werden. [1,5,8]

SOAP Array Attack

Der SOAP Array Angriff zielt auf den Speicher des angegriffenen Systems ab. Hierbei wird eine XML-Datei gesendet, welche ein Array mit einer sehr großen Anzahl an Elementen enthält. Im Arbeitsspeicher des Opfersystems wird anschließend für die Bearbeitung dieses Arrays der benötigte Speicherplatz reserviert. Ist der reservierte Speicherplatz zu groß, wird das System zum Auslagern aus dem Arbeitsspeicher auf die Festplatte gezwungen. Durch dieses Verhalten wird die Performance des angegriffenen Systems enorm eingeschränkt. [1,5,8]

XML Attribute Count Attack

Bei dem XML Attribute Count Angriff wird ein ähnliches Schema wie bei den SOAP Array Attacks angewendet. Die Anzahl der Attribute innerhalb eines Elements wird so weit erhöht, bis der Server an sein Speicherlimit gerät und zum Auslagern der Daten gezwungen wird. Die Performance des angegriffenen Systems sinkt hier ebenfalls enorm.[1,5,8]

XML Element Count Attack

Der XML Element Count Angriff funktioniert analog zum XML Attribute Count Attack. Allerdings wird in diesem Angriff die Anzahl der Elemente, und nicht deren Attribute für die Speicherbelastung, verwendet. Sollte die Anzahl der Attribute eines Elements beschränkt sein, so kann man die Anzahl der Elemente erhöhen, was ebenfalls zu einer erhöhten Speicherbelastung führt.[1,5,8]

XML Entity Expansion Attack

Über XML Entity Expansions können Entitäten innerhalb einer XML-Datei definiert werden. Hierbei kommt Document Type Definition (DTD) zum Einsatz. Ziel des Angriffs ist ebenfalls die Ressource Speicher, was sich auf Performance und Stabilität des Systems auswirken kann.[1,5,8]

XML External Entity Attack

Bei diesem Angriff wird ebenfalls die Möglichkeit der Entitätendefinition verwendet. Im Gegensatz zu der Entitiy Expansion Attack werden hier jedoch externe Entitäten referenziert. Hierbei kann jede beliebige Datenquelle verwendet werden, welche anschließend in den Arbeitsspeicher geladen werden soll. Bei einer sehr großen oder endlosen Datenquelle kann der verfügbare Speicher blockiert werden mit daraus resultierendem Performanceverlust.[1,5,8]

Hash Collision Attack

Da auf einem Server die im Speicher geladenen Elemente schnell zur Verfügung stehen sollen, wird über jedes Element ein Hashwert gebildet und in einer Tabelle gespeichert. Wenn zwei Elemente denselben Hashwert erzeugen, kommt es zu einer Kollision, welche gelöst werden muss. Bei einer solchen Kollision wird enorm mehr Rechenzeit in Anspruch genommen, was sich negativ auf die Performance auswirkt. Im

schlimmsten Fall kann das System die vielen Kollisionen nicht mehr verarbeiten.[1,5,8]

Wie bereits in der Einleitung erwähnt, zielen diese Angriffe auf den Web Service selbst ab und bewegen sich hiermit auf dem ISO/OSI Modell auf Layer 5-7. Natürlich gibt es Angriffe auf den tiefer liegenden Schichten wie zum Beispiel TCP-Syn-Floods, welche auch zu den Denial of Service Angriffen zählen, jedoch das angegriffene System auf Layer 4 attackieren.

III. Aufbau der Testumgebung

Die Testumgebung besteht aus zwei virtualisierten Maschinen (VM): einem Angreifer und einem Server. Die VMs wurden mit dem Betriebssystem Xubuntu installiert, einer Linux Distribution, die mit einer leichtgewichtigen grafischen Oberfläche (GUI), welche auf VMs performant und stabil läuft. Auf dem Server wurden zwei voneinander getrennte Tomcat Application Server installiert. Auf diesen werden die Web Services deployed. Es wurden mehrere Application Server Instanzen verwendet, um die verwendeten Frameworks (Axis2 und CXF) voneinander zu trennen und die möglichen, damit verbundenen Kompatibilitätsprobleme schon im Vorfeld auszuschließen. Damit die Server parallel gestartet werden können, wurden die Listener auf unterschiedlichen TCP-Ports konfiguriert.

Wie in Tabelle 1 zu sehen ist, verfügt der Angreifer über halb so viel zugewiesenen Ressourcen wie der Server. Diese Zuweisung wurde so gewählt, damit es sich schnell bemerkbar macht, wenn der Aufwand eines Angriffes beim Angreifer größer ist als das erzielte Ergebnis am Server. Dieses Verhältnis ist ausschlaggebend, wie effizient ein Angriff ist. Angreifern kommt es natürlich entgegen, wenn der aufgebrachte Aufwand im Vergleich zu dem angerichteten Schaden sehr gering ist. In diesem Fall war die Dimensionierung der Hardware während der Angriffe mehr als ausreichend.

A. *Server*

Für die Durchführung der DoS Angriffe mussten Test Web Services erstellt werden. Die Funktionen hinter diesen Services sollten möglichst leichtgewichtig sein, da die Angriffe auf die Web Services selbst und nicht auf die für die Tests erstellten Funktionen zielen sollen.

Die meisten Angriffsszenarien konnten an einem einfachen Ping Service getestet werden. Dabei ruft der Client die Operation „ping“ des Services auf. Der Service antwortet mit den String „pong“. Dieser Service erzeugt weder viel Netzwerkverkehr, noch ist er ressourcenintensiv was CPU oder Speicherauslastung auf dem Server angeht. Daher kann bei einem positiven Test mit einem der Angriffsszenarien davon ausgegangen werden, dass eine Komponente des Services die Überlast verursacht hat.

Für dem Soap-Array Angriff, welcher unter Kapitel II beschrieben wurde, muss für den Angriff eine Operation vorhanden sein bei der ein Array als Parameter übergeben werden kann. Für diesen Angriffsfall wurde deshalb eine weitere Operation erstellt, welche ein String Array entgegen nimmt und die Anzahl der übergebenen Strings zurückgibt.

Die Web Services wurde mit Eclipse Luna entwickelt. Zu diesem Zweck wurden die Frameworks in die Entwicklungsumgebung eingebunden. Auch hier wurden die Umgebungen strikt voneinander getrennt um gegenseitige Konflikte zu vermeiden. Entwickelt wurden die Web Services nach dem Code-First verfahren. Dabei wurden die eben beschriebenen Funktionalitäten in eine Java Klasse implementiert. Unter Zuhilfenahme des jeweiligen Frameworks wurde auf Basis dieser Java Klasse, das für den Web Service notwendige WSDL File und der dazugehörige Service generiert. Dieses wurde nach der Generierung nicht mehr verändert. Es sollte vermieden werden, dass zusätzliche Sicherheitslücken durch eine manuelle Änderung entstehen. Im Umkehrschluss sollten Fehler bei der Generierung, die zu einer Schwachstelle führen, bei der Auswertung der Tests auffallen.

Tabelle 1: Hardwareausstattung der Testumgebung

	CPU Kerne	Ram
Angreifer	2	4 Gb
Server	4	8 Gb

Die folgenden zwei Frameworks wurden ausgewählt, da Sie die meist verbreitetsten Frameworks für SOAP Services sind.

1) *Axis2 Framework*

Axis2 war ein weit verbreitetes Framework von Apache. Dieses galt lange Zeit als das "[...]de facto open source Java Web Service framework"[3] . Die aktuellste Version zum Zeitpunkt der Arbeitserstellung ist die Version 1.6.2 vom 17.04.2012. Auch wenn diese Version etwas veraltet scheint, sind im Internet immer noch Services zu finden, deren URL auf die Verwendung dieses Frameworks hindeutet. Es ist davon auszugehen, dass dieses bei bestehenden Frameworks noch häufig im Einsatz ist.

2) *CXF Framework*

Das CXF Framework ist ein Open-Source Services Framework, ebenfalls von Apache, welches ein einfach zu bedienendes, auf Standards basierendes Programmiermodell zur Erstellung von Web Services bietet[2]. In dieser Arbeit wird die Version 3.0.3 eingesetzt, welche am 11.12.2014 veröffentlicht wurde. Dies ist damit das aktuellste Framework, welches in dieser Arbeit getestet wird.

B. *Angreifer*

Der Angreifer wurde mit der Software WS-Attacker ausgestattet. Diese Software stellt die eingesetzten Angriffe per Plugins zur Verfügung und ist durch diesen modularen Aufbau mit weiteren Plugins erweiterbar.

Jeder Angriff durch diese Software wird mit vorkonfigurierten Parametern bereit gestellt. Diese Konstellation der Parameter wurde in den anfänglichen Tests verwendet, da diese schon in der Lage sind, aussagekräftige Ergebnisse zu liefern. Um diese Ergebnisse zusätzlich zu validieren wurde die WSDL Datei per Web Browser in regelmäßigen Abständen abgerufen. Wenn diese Datei nicht mehr abgerufen werden kann, kann von einem erfolgreichen Denial of Service Angriff ausgegangen werden

IV. Testdurchführung

Die Tests wurden unter Laborbedingungen in einem abgekapselten, geschlossenen Netzwerk mit möglichst geringen störenden äußeren Einflüssen durchgeführt. Hierbei wurde

nach jedem Angriff dem Server genug Zeit gelassen, damit sich dieser wieder erholen konnte. Vor jeder Testserie wurde zudem das System neu gestartet, damit eventuelle Altlasten aus dem Speicher entfernt werden. Diese Vorkehrungen stellten sicher, dass weitestgehend möglichst unabhängige Testergebnisse erzielt wurden.

A. *Definition der Messgrößen*

Der Zustand des Servers wurde während der Angriffe über einige Faktoren überwacht. Anfänglich wurde dessen Erreichbarkeit per ICMP-Ping über das Netzwerk analysiert. Allerdings waren diese Messwerte nicht aussagekräftig, da selbst bei sehr hoher Last und einem erfolgreichen Denial of Service Angriff der Server in der Regel unter einer Millisekunde geantwortet hat. Hieran ist zu sehen, dass das Überwachen durch klassische Messgrößen zu keinem befriedigendem Ergebnis geführt hat.

Daher wird in dieser Arbeit zusätzlich auch die Unix Load überwacht. Dies ist eine positive Zahl, welche vom System ausgegeben wird. Diese Zahl geht gegen Null, wenn das System sehr wenig Last ausgesetzt ist. In diesem Fall stehen fast die kompletten Hardwareressourcen für weitere Berechnungen zur Verfügung. Wächst diese Zahl, steht das System zunehmend unter einer größeren Last. Zum Beispiel ist bei einer Load von 2,0 die betroffene, schwächste beteiligte Komponente im System um 100% überbucht. Falls diese enorm hohe Werte erreicht, kann man von einem erfolgreichen Denial of Service ausgehen. In den durchgeführten Tests wurde nicht nur die Antwortzeit des Services überwacht, sondern ebenfalls die Load. An diesen beiden Werten kann man den Erfolg eines Angriffes messen, da ein Server mit einer Load von über 135 sehr unwahrscheinlich in seinen normalen Betrieb zurück findet. Während dieser Testreihe wurde ein Server, welcher nach dem Angriff eine Load von ~115 hatte, weitere 24h Zeit gegeben, um seine Aufgaben abzuarbeiten und seine Load zu normalisieren. Nach den 24h war die Load des Servers auf über 200 angewachsen und er war nicht mehr zu gebrauchen, obwohl es keinen weiteren Angriff gab. Hierbei ist zu erkennen, dass einige der Angriffe enorme Nachwirkungen auf den Servern verursachen.

B. *Durchführung der Tests*

Die Tests mit den genannten Angriffen wurden in zwei Phasen durchgeführt. In der ersten Phase wurden die Angriffe mit den von den Entwicklern des WS-Attacker vorgeschlagenen Standardparametern durchgeführt. In der zweiten Phase wurden die in der vorherigen Phase entdeckten Schwachstellen gezielt mit höherem Aufwand angegriffen, um deren Auswirkungen zu analysieren. Im folgenden werden nun die Parameter behandelt, welche in der zweiten Testreihe verändert wurden. In der Regel wird mit zwei parallelen Threads der Angriff durchgeführt. Bei jedem Angriff werden pro Thread vier Anfragen an den Server gestellt. Mit einem weiteren Parameter kann die jeweilige Payload pro Angriff eingestellt werden. Unter dem Begriff der Payload wird die Größe der manipulierten Daten pro Anfrage definiert. Diese drei Werte wurden in der zweiten Testreihe um den Faktor von vier vergrößert.

C. *Auswertung der Messergebnisse*

Wie in Tabelle 2 zu erkennen ist, spricht Axis2 auf einige der Angriffe in der ersten Phase der Tests an. Der erste Testdurchlauf wurde Basic genannt, da hier alle Angriffe auf den Web Service getestet werden. Im zweiten Durchlauf, Advanced genannt, wurde mit, angepassten Parametern erneut getestet, wie unter Durchführung der Tests beschrieben. Hierbei wurden nur Angriffe durchgeführt, welche in dem vorhergehenden Testdurchlauf mit einem positiven Ergebnis (✘) abgeschlossen hatten. Testergebnisse für Angriffe, welche in der ersten Runde fälschlicherweise als erfolgreich notiert wurden und hier fehlschlugen wurden konsolidiert, da der Angriff zu keinem weiteren Schaden am System geführt hat. Die Angriffe, welche richtig erkannt wurden, hinterließen aller-dings erheblich geschädigte Systeme, die in der Regel nur noch durch einen Neustart zu retten waren. Wie ebenfalls in Tabelle 2 zu erkennen ist, wurden beide Web Services getrennt voneinander getestet - ohne Unterschiede bei den Ergebnissen .

Tabelle 2: Testergebnisse Axis2

Test	Ping Basic	Ping Advanced	Array Basic	Array Advanced
Coercive Parsing	✔	✔	✔	✔
DJBX31A Hash Collision Attack	✔	✔	✔	✔
DJBX33A Hash Collision Attack	✔	✘	✔	✘
DJBX33X Hash Collision Attack		✘	✘	✘
SOAP Array Attack	✘	✘	✘	✘
XML Attribute Count Attack	✔	✔	✔	✔
XML Element Count Attack	✘	✘	✘	✘
XML Entitiy Expansion (recursive)	✘	✘	✘	✘
XML External Entitiy Attack	✘	✘	✘	✘
XML Overlong Names Attack	✘	✘	✘	✘

Zum Vergleich wurden dieselben Tests in identischen Basic und Advanced Testreihen mit dem CXF basierten Web Service durchgeführt. Wie in Tabelle 3 zu erkennen ist, wurde lediglich ein Service fälschlicherweise als positiv erkannt. In der zweiten Testreihe wurde der Angriff durchgeführt, ohne dass dieser beim Server zu negativen Auswirkungen führte. Der Web Service mit CXF Framework hat sich in den Tests als sehr robust und höchst performant durchgesetzt. Selbst nach größeren Angriffswellen hatte sich die Last auf dem Server wieder normalisiert, ohne dass sein Dienst zu irgend einem Zeitpunkt beeinträchtigt war.

D. *Historischer Vergleich der Messergebnisse*

Wie in der Veröffentlichung [1] unter *TABLE V* zu sehen ist, wurden diese Probleme bei Axis2 bereits im Jahre 2013 gefunden. Zum damaligen Zeitpunkt hatte das CXF Framework ebenfalls vergleichbare Probleme. Zum Zeitpunkt dieser Veröffentlichung hatte allerdings das Entwicklerteam von CXF alle Probleme beseitigt. Bei Axis2 hat sich nichts getan - die Angriffe, welche 2013 zu Problemen geführt haben, führen heute ebenfalls zum selben Ergebnis.

Tabelle 3: Testergebnisse CXF

Test	Ping Basic	Ping Advanced	Array Basic	Array Advanced
Coercive Parsing	✘	✘	✘	✘
DJBX31A Hash Collision Attack	✘	✘	✘	✘
DJBX33A Hash Collision Attack	✘	✘	✘	✘
DJBX33X Hash Collision Attack	✘	✘	✘	✘
SOAP Array Attack	✘	✘	✘	✘
XML Attribute Count Attack	✘	✘	✘	✘
XML Element Count Attack	✘	✘	✘	✘
XML Entitiy Expansion (recursive)	✘	✘	✔	✘
XML External Entitiy Attack	✘	✘	✘	✘
XML Overlong Names Attack	✘	✘	✘	✘

V. Lösungsansätze / Future Work

A. *Abhärtung der XML/Parser / Wechsel des Frameworks*

Im Vergleich ist bei den Testergebnis ein deutlicher Unterschied zwischen den beiden getesteten Frameworks zu erkennen. Beim Axis2 Framework ist keine Verbesserung zu erkennen. Hier liegt die Vermutung nahe, dass der eingesetzte XML Parser mit den Angriffen überfordert ist. Bei der Auswahl des Frameworks sollte also der Security Aspekt berücksichtigt werden. In dieser Arbeit wurden Web Services verwendet, welche aus den Java Klassen generiert wurden. In diesem Beispiel wäre ein Wechsel des Frameworks ohne größeren Aufwand möglich. Ob dies bei komplexeren Services in der Praxis ebenso möglich ist, müsste im Einzelfall evaluiert werden. Manuelles Härten des Web Services gegen Angriffe dieser Art wird in der Arbeit von *Ludwig Stage* [5] behandelt, wobei dies für uns keine Lösung des Problems darstellt. Hierbei wird lediglich ein Weg des Angriffs eingeschränkt, allerdings nicht die Ursache. Bei ausreichenden einzelnen Anfragen durch Verteilung der Angreifer oder einer weiteren Angriffstechnik, welche auf dieselbe Schwachstelle abzielt, würde der Angriff erneut zum gewünschten Erfolg führen.

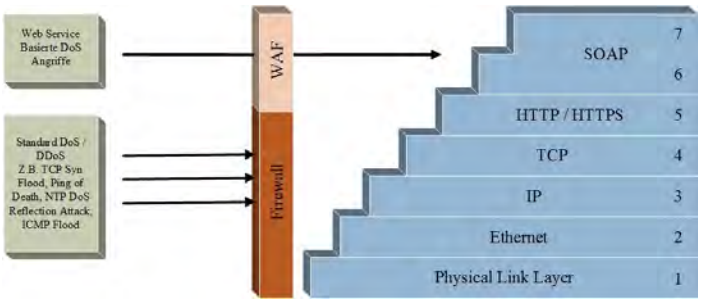


Abbildung 1: Funktionsweise WAF Firewalls - ISO/OSI Modell

B. *Verteidigung mit Firewall Systemen*

Firewall-Systeme sind der zentrale Punkt für den Schutz von Netzwerken, Servern und anderen Systemen. Die Firewall überprüft dabei jedes Netzwerkpaket und entscheidet anhand von Regeln ob, dieses weitergeleitet werden soll oder nicht. Firewall-Systeme haben sich in den letzten Jahren stark verändert. So waren die ersten Firewalls einfache Paketfilter, welche jedes einzelne Paket anhand eines Regelwerks geprüft haben. Die nächste Generation von Firewall-Systemen war bereits in der Lage, Session basiert die Regeln anzuwenden. Dies hatte erhebliche Performanzvorteile, da die Firewall nur noch das erste Paket einer Session überprüfen und mit den Regeln vergleichen musste. Die restlichen, zu der selben Session gehörenden Pakete konnten ohne weitere Überprüfung weitergeleitet werden. Dies war möglich, da die Firewall über eine Session-Tabelle verfügte, in welche die erlaubten Sessions eingetragen wurden. Diese Firewalls nennen sich Stateful Firewalls. In Abbildung 1 wird dargestellt, auf welchen Schichten des ISO/OSI Modells diese Firewalls Schutz vor DoS Angriffen bieten.

Mit zunehmender Kreativität der Angreifer wurden allerdings auch die Firewalls intelligenter, so dass sie heute in der Lage sind, eine Vielzahl von Angriffsmustern zu erkennen. Allerdings hat dies auch seinen Preis. Die Überprüfung von Netzwerkverkehr auf verschiedene Angriffsmuster erfordert, dass die Firewall tiefer in die Pakete hineinschaut. Das bedeutet, dass die Firewall die Pakete nicht nur bis Schicht 3 oder 4 des ISO/OSI Modells analysieren muss, sondern auch die darüber liegenden Schichten. In diesem Zusammenhang wird die Gefahr der SOAP basierten DoS Angriffe in Abbildung 1 veranschaulicht. Die SOAP basierten DoS Angriffe erscheinen dabei für die Firewall als legitime Anfragen. Firewalls, welche in der Lage sind, auch auf den Schichten 5-7 die Pakete zu untersuchen, werden als Web Application Firewalls (WAF), oder Deep Paket Inspection Firewalls, bezeichnet.

Die Firma Checkpoint brachte 2002 ein Update für ihre Firewall Systeme mit folgenden Statement: "Checkpoint [...] has enhanced its market-leading Internet security solution [...] to provide integrated security for Web services' two most commonly used protocols, XML and SOAP - making Check Point the first and only company to provide complete XML and SOAP security [...][6]". Demnach gibt es seit 2002 Web Application Firewalls, welche auch SOAP Web Services absichern können. Unklar ist jedoch weiterhin ob diese Firewalls auch vor Web Service basierten DoS Angriffen Schutz bieten.

Allerdings haben diese zusätzlichen Sicherheitsfunktionen auch ihren Preis. Die Deep Packet Inspection kostet Performance. Die Sessions, welche auf diese Weise geprüft werden sollen, müssen wieder vollständig analysiert werden. Hier fällt also der oben erwähnte Performanzvorteil weg, der durch die Stateful-Firewalls entstanden ist. Weiterhin müssen neue Komponenten in die Firewalls integriert werden um die zusätzlichen Sicherheitsfunktionen abzubilden. Diese Veränderungen wirken sich auf die Firewall-Performance aus und müssen mit zusätzlicher Hardware ausgeglichen werden.

C. *Zusammenfassung*

Es wurden verschiedene DoS Angriffe auf Webservices untersucht und Möglichkeiten beschrieben, Webservices vor DoS Angriffen zu schützen. Die richtige Wahl

des Frameworks scheint eine der effizientesten und kostengünstigsten zu sein. Hier ist das Apache CXF Framework zu bevorzugen, da dieses durch keine der untersuchten Methoden angreifbar ist. Inwieweit Firewalls diesen Schutz bei anderen Frameworks übernehmen können ist unklar. Wahrscheinlich sind aktutelle Firewalllösungen in der Lage entsprechenden Angriffsmuster zu erkennen. Sicher ist, dass die Firewall-Lösung die kostenaufwendigste ist. Die in [5] erwähnten Modifikationen der Web Services bieten eventuell ebenfalls Schutz.

REFERENCES

[1] A. Falkenberg, C. Mainka, J. Somorovsky, and J. Schwenk, "A New Approach towards DoS Penetration Testing on Web Services," in 2013 IEEE International Conference on Web Services (ICWS), pp. 491–498.

[2] N. Balani and R. Hathi, Apache CXF web service development: Develop and deploy SOAP and RESTful web services. Birmingham, U.K.: Packt Pub, 2009.

[3] D. Jayasinghe and A. Azeez, Apache Axis2 web services: Create secure, reliable, and easy-to-use web services using Apache Axis2, 2nd ed. Birmingham, U.K.: Packt Pub, 2011.

[4] Hugo Haas, Allen Brown, Web Services Glossary: W3C Working Group Note 11 February 2004. Available: http://www.w3.org/TR/ws-gloss/.

[5] Ludwig Stage, "Entwurf einer Methodik zum Testen der Sicherheit von Web-Service-basierten Systemen," Diplomarbeit, Institut für Architektur von Anwendungssystemen, Universität Stuttgart, Stuttgart, 2013.

[6] Check Point Software First to Secure Web Services: XML/SOAP security made possible with Breakthrough Application Intelligence Technology. Available: http://www.checkpoint.com/press/2002/xmlsoap082702.html .

[7] WS-Attacker - Project Web Site. Available: http://ws-attacker.sourceforge.net.

[8] M. Spreitzer and J. d. Meer, Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference: WSFAggressor: An Extensible Web Service Framework Attacking Tool. Montreal, Québec, Canada, 2012.

[9] Information technology - Open Systems Interconnection - Basic Reference Model: TheBasic Model, 7498-1, 1994.

Anwendung von linearen Klassifizierern zur Kategorisierung von Texten

Klassifizierung mittels linearen Support Vector Machines (SVM)

Martin Haschke
Informatik Master
Hochschule Furtwangen University
martin.haschke@hs-furtwangen.de

Aiko Isselhard
Informatik Master
Hochschule Furtwangen University
aiko.isselhard@hs-furtwangen.de

***Abstract*— In dieser Ausarbeitung wird die Funktionsweise, Eignung und die Performance von Support Vector Machines (SVM) zur Textkategorisierung betrachtet. Hierzu wurden verschiedene Arten von Tests mit verschiedenen Metriken und Datensätzen zusammengetragen, um einen objektiven Eindruck von SVM's zu erhalten. Ein wichtiges Merkmal der Tests ist eine gemeinsame Basis, um mehrere, verschiedene Klassifizierungsalgorithmen miteinander vergleichen zu können.**

In der Ausarbeitung wurde der Schwerpunkt auf eine einfach zu verstehende Formulierung, grafische Darstellung und Erklärung gelegt, um das Verständnis zu fördern. Technische Details, sowie formale Beweise zu den hier genannten Vorteilen und Eigenschaften von SVM's können in den Quellen nachgelesen werden. Zudem wurde diese Ausarbeitung auf linear-separierbare Texte beschränkt, damit die Funktionsweise leichter zu verstehen ist.

Keywords—SVM, Support Vector Machine, Text Kategorisierung, Lineare Separierbarkeit von Textklassen, Invarianztransformation, Precision/Recall-Breakeven-Point

I. Einleitung

Seitdem lineare Klassifizierer für Suchmaschinen eingesetzt werden, konnten auf diesem Gebiet sehr große Erfolge verzeichnen werden. Die Usability ist der von klassischen Suchen mit booleschen Ausdrücken sehr weit überlegen. Allerdings gibt es bei den linearen Klassifizierern mehr als einen passenden Algorithmus, mit unterschiedlichen Stärken und Schwächen. Je nach Anwendungsfall muss der richtige Algorithmus gewählt werden. Allerdings scheinen die SVM's hier in jeglicher Hinsicht die besten Ergebnisse zu liefern, was in dieser Ausarbeitung näher beleuchtet wird.

Grundlagen zu der Funktionsweise von linearen Klassifizierern bei der Textklassifizikation können in Kapitel II nachgelesen werden. Die Funktionsweise und strukturellen Vorteilen von SVM's können sind in Kapitel III aufgeführt. In Kapitel IV und V wurden Parameter und Tests zusammengetragen, welche die Stärken und Vorteile von SVM's gegenüber vergleichbaren linearen Klassifizierungsalgorithmen verdeutlichen. In Kapitel VI wird ein abschließendes Fazit über die zusammengetragenen Informationen gezogen. Ebenfalls sind dort Möglichkeiten für weitere Ausarbeitungen und Forschungen zu finden.

II. Grundlagen

In den folgenden Abschnitten geht es um die Funktionsweise von Support Vektor Machines und deren Anwendung zur Klassifizierung von Texten. Um einen Text zu klassifizieren, wird dieser in einen Wort-Vektor, auch Feature-Vektor genannt, konvertiert. Dabei entspricht jedes Wort einer Dimension in diesem Feature-Vektor. Der Wert eines Elements des Vektors entspricht der Anzahl eines Wortes, welches in dem zu klassifizierenden Text vorkommt. Enthält ein Text beispielsweise fünfmal das Wort „und", hat der entsprechende Vektor in der „und"-Dimension den Wert Fünf. Durch die Umwandlung in einen Feature-Vektor können alle Texte in einem n-dimensionalen Raum dargestellt werden. Dadurch können mathematische Funktionen und Algorithmen auf die Feature-Vektoren angewandt werden.

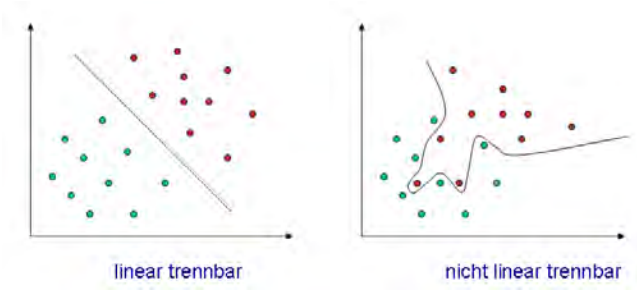


Abbildung 1. Lineare Trenn- bzw. Separierbarkeit [1]

Wie bereits erwähnt, werden in dieser Ausarbeitung nur linear-separierbare Klassen betrachtet. Eine Klasse ist gleich zu setzen mit einer Kategorie von Texten. In der Abbildung 1 wurden zweidimensionale Feature-Vektoren eingezeichnet. Jeder Feature-Vektor entspricht einem Text, dessen Kategorie bekannt ist. Die Texte wurden dabei in zwei Kategorien, gekennzeichnet durch Rot und Grün, separiert. Jeder Text kann in diesem Beispiel nur einer Kategorie angehören. Ein Problem ist linear Trennbar, wenn sich alle Feature-Vektoren so durch eine Gerade trennen lassen, dass alle Vektoren auf der einen Seite der Geraden zu derselben Kategorie und alle Feature-Vektoren auf der anderen Seite der Geraden zur anderen Kategorie gehören. Haben die Feature-Vektoren n Dimensionen, entspricht die Gerade einer n-dimensionalen Hyperebene [2].

In vielen Klassifizierungs-Problemen der realen Welt ist lineare-separierbarkeit nicht gegeben. Joachims [3] schlägt jedoch vor, dass dies für die Text-Kategorisierung nicht zutrifft. So sind beispielsweise alle Kategorien des bekannten Ohsumed-Datensatzes [2][4], sowie die meisten Kategorien des Reuters-Datensatzes [5][4], linear separierbar.

III. SVM - Support Vector Machine

Die Support Vector Machines wurden auf der COLT-92 von Boser, Guyon und Vapnik erstmalig vorgestellt.

SVMs müssen vor ihrem Einsatz trainiert werden. Dazu werden Trainingsdaten benötigt. Die Trainingsdaten werden durch Feature-Vektoren repräsentiert. Für jeden Vektor muss die zugehörige Klasse bekannt sein. Im Beispiel in Abbildung 2 wurden mehrere zweidimensionale Feature-Vektoren in zwei Klassen unterteilt: Klasse 1 entspricht roten Quadraten, Klasse 2 blauen Kreisen. Die Feature-Vektoren wurden dabei so gewählt, dass diese linear-trennbar sind.

Die Abbildung 2 zeigt zwei mögliche Trenngeraden, also Hyperebenen im zweidimensionalen Raum, mit verschiedenen Randgrößen. Diese Randgröße, in Hellblau eingezeichnet, entspricht dem Abstand der Gerade zu dem Feature-Vektor, der ihr am nächsten ist. Um neue, noch unbekannte Feature-Vektoren zu klassifizieren, wird überprüft, auf welcher Seite der Trenngeraden sie sich befinden. Befinden sie sich oberhalb der Trenngerade, gehören sie Klasse 1 an, liegen sie Unterhalb, werden sie der Klasse 2 zugeordnet.

Angenommen, wir wollen den in Grün eingezeichneten Feature-Vektor klassifizieren, würde er nach der Trenngeraden B in Klasse 2 gehören. Aufgrund der gegebenen Testdaten ist es wahrscheinlicher, dass er in Klasse 1, zu den roten Quadraten gehört. Trenngerade B hat demnach eine höhere Wahrscheinlichkeit, neue Dokumente falsch zu klassifizieren. Statistisch gesehen klassifiziert eine Trenngerade mit einem breiten Randbereich neue Dokumente mit einem kleineren Fehler richtig. Das heißt Gerade A, die Gerade mit dem maximal breiten Randbereich, ist die Gerade, welche neue Feature-Vektoren mit dem kleinsten möglichen Fehler richtig klassifiziert.

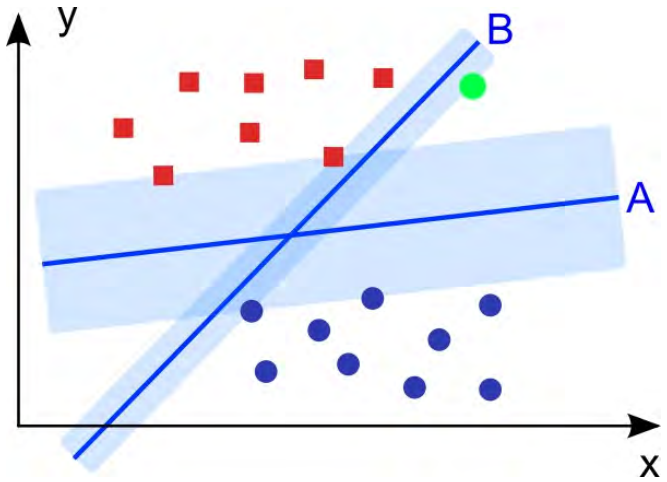


Abbildung 2. Feature Vektoren im Zweidimensionalen Raum [6]

Aufgabe der SVM ist es, die Klassen durch eine Hyperebene zu trennen. Dabei wird die Hyperebene so gewählt, dass der Abstand der Vektoren, die der Ebene am nächsten sind, maximiert wird. Dieser möglichst breite Rand sorgt später dafür, dass neue, noch nicht klassifizierte Vektoren, in die richtige Klasse sortiert werden. „Richtig“ bedeutet in diesem Fall, dass der Vektor mit der kleinst-möglichen Wahrscheinlichkeit falsch einsortiert wird.

Um diese Gerade, bzw Hyperebene festzulegen, müssen nicht alle Vektoren betrachtet werden. Vektoren, die von der Hyperebene weiter weg und hinter anderen Vektoren liegen, beeinflussen die Lage der Hyperebene nicht. Die Position der Hyperebene hängt nur von den ihr am nächsten liegenden Vektoren ab. Diese nennt man Stützvektoren. Die Support-Vector-Machine, zu Deutsch Stützvektormaschine, verdankt diesen Stützvektoren ihren Namen.

SVMs sind im allgemeinen auch in der Lage, mit nicht linear trennbaren Klassen umzugehen. Zu diesem Zweck werden sogannte Kerne verwendet. Dies wird in diesem Paper jedoch nicht betrachtet.

Die Einsatzgebiete von SVMs sind so groß wie unterschiedlich. SVMs werden in vielen Bereichen wie Machine Learning, Neuronalen Netzen, Statistischen Analysen etc. genutzt und es gibt viele kommerziellen und nicht kommerzielle Umsetzungen von SVMs. Bekannte Umsetzungen sind unter anderem die LibSVM (C++), SVMLight (C), Torch (C++), Spider (Matlab) und Weka (Java).

IV. SVMs in der Text-Klassifizierung

Die Feature-Vektoren von Text-Dokumenten können mitunter sehr viele Dimensionen haben. Eine Grundvoraussetzung für einen Algorithmus zur Text-Kategorisierung ist also, dass dieser in der Lage ist, mit großen Feature-Vektoren zu arbeiten. Hat ein Modell viele Parameter, aber nur wenige Testdaten, besteht immer die Gefahr des Overfitting - zu Deutsch der Überanpassung. Durch die großen Feature-Vektoren ist es in der Text-Kategorisierung oft der Fall, dass mehr Dimensionen, also Parameter, als Test-Daten vorhanden sind.

Wenn nun 2 Hyperebenen h und h’ existieren, nennt man die Hyperebene h überangepasst, wenn

1) h einen kleinen Fehler gegenüber den Trainingsdaten hat als h’
2) h’ aber einen kleinen Fehler gegenüber allen Daten hat (vor allem nicht Trainingsdaten).

Der Fehler sagt in diesem Fall aus, wie viele Dokumente falsch einsortiert werden. Die Hyperebene h’ entspricht der „perfekten“ Hyperebene, welche alle Dokumente mit dem kleinstmöglichen Fehler einsortiert. Die Hyperebene h’ ist hypothetisch, da um h’ zu bestimmen, die Trainingsdaten alle existierenden Dokumente enthalten müsste.

Die Hyperebene h erhält man aus einer begrenzten Menge von Trainingsdaten. Überanpassung nennt man den Zustand, wenn die „perfekte“ Hyperebene h’ bei der Kategorisierung der Trainingsdaten, mit denen die Hyperebene h bestimmt wurde, mehr Fehler macht als die Hyperebene h. Das heißt, durch ungünstige oder zu wenige Trainingsdaten erhält man

eine Hyperebene, welche einen scheinbar kleinen Kategorisierungsfehler hat. Je kleiner die Menge der Trainingsdaten und je größer die Anzahl der Features ist, desto größer ist die Gefahr der Überanpassung.

In diesem Fall wäre der Klassifizierungsalgorithmus zu spezialisiert. Er erkennt zwar die gelernten Trainings-Daten richtig, ordnet aber viele neue Daten falsch ein.

Für statistische Klassifizierungsalgorithmen gilt die Vapnik-Chervonenkis-Dimension, kurz VC-Dimension, als Maß für die Komplexität. Im Falle der SVM korreliert die VC-Dimension mit der Anzahl der Dimensionen der Hyperebene. SVMs haben eine „Overfitting Protection“. Die „Structural Risk Minimization“, ein mathematisches Prinzip, das der SVM zugrunde liegt, sorgt dafür, dass die VC-Dimension nicht zu groß wird. Dies bedeutet, dass SVMs ohne Probleme mit Vektoren mit sehr vielen Features arbeiten können.

Obwohl es in der natürlichen Sprache viele, auf den ersten Blick unnötige Wörter und Füllwörter gibt, zeigt ein Experiment von [Joachims, 1997][2], dass dies nicht der Fall ist. Abbildung 3 zeigt das Ergebnis eines Experiments mit allen Dokumenten der „acq“-Kategorie aus dem Reuters-Datensatz.

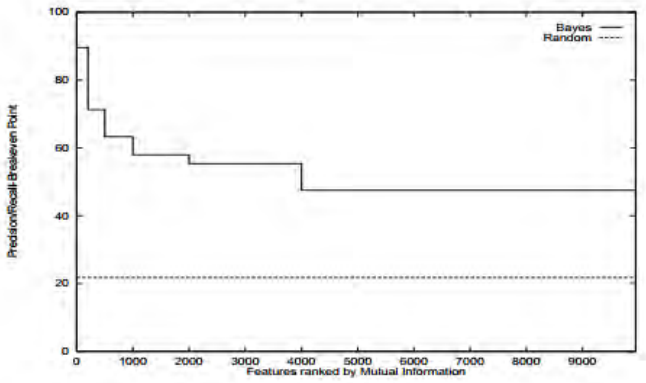


Abbildung 3. Informationsgehalt der Features [2]

Alle Features wurden nach ihrem Informationsgehalt sortiert. Anschließend wurde ein naiver Bayes-Klassifizierer mit den Features trainiert. Dabei wurden nur die Features von Rank 1-200, 201-500, 501-1000, 1001-2000, 2001-4000 und 4001-9962 verwendet. Anschließend wurde jeweils der schlechteste Precision/Recall-Breakeven Point (siehe Kapitel V-B) ausfindig gemacht und in der Abbildung 3 dargestellt. Hierbei ist im Gegensatz zu den Erwartungen zu erkennen, dass selbst anscheinend informationsarme Wörter bei der Textklassifizierung eine erkennbare Rolle spielen.

Die Overfitting-Protection funktioniert auch, wenn die Anzahl der Test-Daten viel kleiner ist, als die Anzahl der Features. Ein Problem, welches in den meisten Problemstellungen der Text-Kategorisierung gegeben sein dürfte. All diese Punkte sprechen für die gute Eignung von SVMs im Bereich der Text-Kategorisierung.

V. Performance

In diesem Abschnitt werden Metriken und Ausarbeitungen behandelt, welche es ermöglichen, die Leistung und Qualität von verschiedenen linearen Klassifizierern miteinander zu vergleichen. Anschließend folgen die Ergebnisse ihrer Ausarbeitung, die die Vor- und Nachteile der SVM's gegenüber den konventionellen Klassifizierern verdeutlichen.

A. Bernhard Scholkopf, Klaus-Robert Müller, Alexander J. Smola, 1999 [7]

Im Jahre 1999 wurde von Bernhard Scholkopf, Klaus-Robert Müller und Alexander J. Smola eine Testreihe durchgeführt, in welcher die Klassifizierungsfehler basierend auf dem US Postal Service Benchmark (USPS) ermittelt wurden. Der USPS besteht aus 7291 Bildern mit einer Auflösung von 16x16 Pixeln, um den Algorithmus auf Handschriftenerkennung zu trainieren. Anschließend stehen weitere 2007 Bilder zur Kategorisierung zur Verfügung, um den Algorithmus zu testen. Wie in Tabelle I zu sehen ist, liegt hier selbst der standard SVM Klassifizierer, eine SVM ohne Kern, schon sehr weit vorne. Unter Verwendung von Invariantstransformationen lässt sich das Ergebnis des SVM Algorithmus noch weiter optimieren.

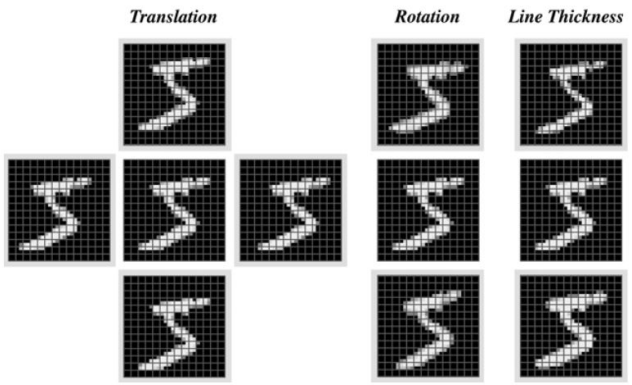


Abbildung 4. Veranschaulichung von Invarianten bei der Handschriftenerkennung [8]

Invariantstransformationen [8] sind mathematische Transformationen der Stützvektoren. Die Stützvektoren auf bestimmte Eigenschaften verallgemeinert, womit die Wahrscheinlichkeit einer korrekten Klassifizierung gesteigert wird. Sehr gut lässt sich dies bei den Transformationen zur Handschriftenerkennung [8] in Abbildung 4 erkennen. Die Grafiken mit grauem Rahmen stellen die virtuellen Stützvektoren dar, welche aus den Stützvektoren (ohne Rahmen) per Transformation erzeugt wurden. Hierbei wurden Eigenschaften wie Position (Translation) des Zeichens, Lage (Rotation) oder Schriftdicke (Line Thickness) modifiziert.

Die Präzision mancher Algorithmen steigt mit der Anzahl der Trainingsdaten. Aus diesem Grund wurde mit dem von NIST zusammengestellten Datensatz, bestehend aus 120.000 Datensätzen (60.000 Trainingssätze, 60.000 Testsätze) eine weitere Testreihe durchgeführt. In dieser konnten die invarianten SVM's sich mit einer Fehlerrate von 0,8% [7] vor dem bisherigen Rekordhalter (Tangent Distance) platzieren. Zudem

Tabelle I. KLASSIFIKATIONSFEHLER IN PROZENT[%] [7]

Algorithmus	Fehler
lineare PCA und lineare SVM (Schölkopf et al. [7])	8,7
k-Nearest Neighbor (Simard et al [9])	5,9
LeNet1 (LeCun et al. [10])	5,0
Regularisierte RBF-Netze (Rätsch [11])	4,1
Kern-PCA und lineare SVM (Schölkopf et al [7])	4,0
SVM (Schölkopf et al [7])	4,0
Invariante SVM (Schölkopf et al [7])	3,0
Boosting (Drucker et al [12])	2,6
Tangent Distance (Simard et al [9])	2,6
Menschliche Fehlerrate	2,5

haben SVM basierte Kategorisierer in folgenden Tests Rekorde erzielt:

- NIST Handschrifterkennung [13]
- MPI Objekterkennung [14][13]
- CERN charm quark tagging [15]
- Santa Fe Zeitreihen Benchmark [16]
- Gesichtsdetektion (vom MIT gehalten [17])
- sowie Reuter Textkategorisierung (Microsoft Research [17])

B. Joachims, 1997 [2]

1997 hat Joachims [2] hat einen Vergleich von verschiedenen Text-Klassifizierungs-Algorithmen vorgestellt. Es wird der Naive Bayes Klassifizierer, der Rocchio-Alghorithmus, ein gewichteter K-Nearest-Neighbour-Algorithmus (k-NN) und der C4.5 Algorithmus mit der Support Vektor Machine verglichen.

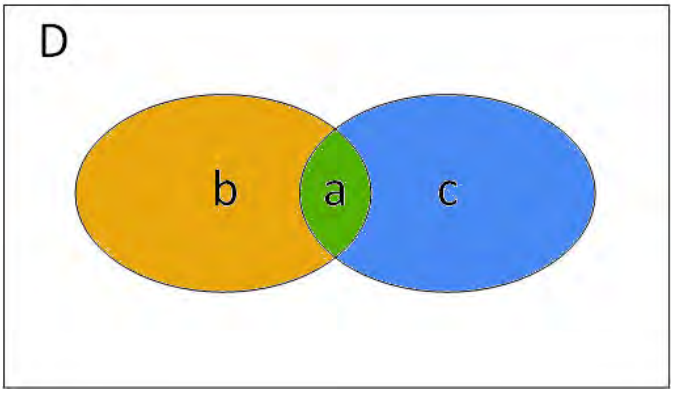


Abbildung 5. Veranschaulichung des Breakeven-Points

Für den Vergleich wird der „ModApte“ Teil des Reuters-21578-Datensatzes [5][4] verwendet. Der „ModApte“-Datensatz wurde von David Lewies kompiliert und enthält 9603-Trainings-Daten und 3299 Test-Daten. Der „ModApte“-Datensatz enthält 135 unterschiedliche Kategorien. Von diesen wurden alle Kategorien gewählt, zu denen mindestens ein Trainings- und ein Test-Datensatz existiert. Dadurch erhält man insgesamt 90 linear separierbare Kategorien.

Nach der Vorverarbeitung enthalten die Trainings-Daten 9962 unterschiedliche Terme. Dies bedeutet, dass die verwendeten Feature-Vektoren bis zu 9962 Dimensionen haben werden.

Als Kennzahl für die Leistungsfähigkeit der Algorithmen wurde der Precision/Recall-Breakeven Point gewählt. Der Breakeven Point ist dann erreicht, wenn der Precision-Wert gleich dem Recall-Wert ist. Eine Query findet N Dokumente. Dann entspricht a der Anzahl der gefunden, relevanten Dokumente. b entspricht der Anzahl der relevanten Dokumente, die nicht gefunden wurden. c entspricht der Zahl der nicht-relevanten Dokumenten die gefunden wurden. Dann entspricht $N = a + c$. R (Recall) und P (Precision) entsprechen in Gleichung 1, bzw. 2 angegeben Formel.

$$P = \frac{a}{a+b} \tag{1}$$

$$R = \frac{a}{a+c} \tag{2}$$

Dann enstpricht der Precision/Recall-Breakeven Point Gleichung 3

$$P = R \tag{3a}$$
$$\frac{a}{a+b} = \frac{a}{a+c} \tag{3b}$$
$$a(a+c) = a(a+b) \tag{3c}$$
$$a^2 + ac = a^2 + ab \tag{3d}$$
$$ac = ab \tag{3e}$$
$$c = b \tag{3f}$$

Der Breakeven Point bezeichnet demnach den Zustand, wenn gleich viele relevante Dokumente (b) nicht gefunden wurden, wie nicht-relevante Dokumente (c) gefunden wurden. In Abbildung 5 wird dies verdeutlicht. Der Breakeven Point ist erreicht, wenn die Mengen von b und c identisch sind. Er ist jedoch unabhängig von der Menge a, der Menge der gefundenen, relevanten Dokumenten.

Jede Kategorie in dem Reuters-Datensatz führt zu einem Recall- und Precision Wert pro Kategorie. Um diese in einer Kennzahl zusammenfassen zu können, wird der Mikro-Durchschnitt gebildet. Um den Mikro-Durschnitt zu bilden, werden die a, b und c-Werte der einzelnen Kategorien wie in Gleichung 4 und 5 addiert.

$$P_{\mu Avg} = \frac{\sum_{i=1}^{90} a_i}{\sum_{i=1}^{90} a_i + \sum_{i=1}^{90} b_i} \tag{4}$$

$$R_{\mu Avg} = \frac{\sum_{i=1}^{90} a_i}{\sum_{i=1}^{90} a_i + \sum_{i=1}^{90} c_i} \tag{5}$$

Damit der Test nicht durch falsche Parameter-Wahl oder andere Einflüsse verfälscht wird, wurde jeder Test mehrfach durchgeführt. Die Features wurden nach ihrem Informationsgehalt sortiert und der Test wurde mit jedem Algorithmus mit den besten 500, den besten 1000, den besten 2000, den besten 5000 und mit allen 9962 Features durchgeführt.

Zusätzlich wurden für jede der Feature-Gruppen der Rocchio Algorithmus jeweils mit einem Wert von $\beta \in \{0, 0.1, 0.25, 0.5, 1.0\}$ und der k-Nearest-Neighbour Algorithmus mit $k \in \{1, 15, 30, 45, 60\}$ durchgeführt. Anschließend wurde für jeden Algorithmus jeweils das beste Ergebnis in die Auswertung übernommen.

Tabelle II. BREAKEVENT-POINT DER 10 AM HÄUFIGSTEN VORKOMMENDEN REUTERS-KATEGORIEN, SOWIE DER MIKRO-DURSCHSCHNITT. [2]

Kategorie	Bayes	Rocchio	C4.5	k-NN	SVM
earn	95,9	96,1	96,1	97,3	98,2
acq	91,5	92,1	85,3	92,0	92,6
money-fx	62,9	97,6	69,4	78,2	66,9
grain	72,5	79,5	89,1	82,2	81,3
crude	81,0	81,5	75,5	85,7	86,0
trade	50,0	77,4	59,2	77,4	69,2
interest	58,0	72,5	49,1	74,0	69,8
ship	78,7	83,1	80,9	79,2	82,0
wheat	60,6	79,4	85,5	79,6	83,1
corn	47,3	62,2	87,7	77,9	86,0
Mikro-Durchschnitt	72,0	79,9	79,4	82,3	84,2

Rocchio, C4.5 und k-Nearest-Neighbour haben jeweils mit der Feature-Gruppe der Besten 1000 Features die besten Ergebnisse geliefert. Rocchio hat mit $\beta = 1,0$ und k-Nearest-Neighbour mit $k = 30$ die besten Ergebnisse geliefert. Der Naive Bayes hat am besten mit allen Features abgeschnitten. Der Versuch zeigt, dass der k-Nearest-Neighbour unter den konventionellen Klassifizierungs-Algorithmen am besten abgeschnitten hat. Trotz dieser Werte hat die SVM, unabhängig von der Parameterwahl, besser als alle verglichenen konventionellen Klassifizierungs-Algorithmen abgeschnitten.

Die SMVs waren mit jeder Feature-Gruppe, ob 500, 1000, 2000, 5000 oder allen Features, besser als alle konventionellen Algorithmen. Die SVM hat den besten konventionellen Klassifizierer in 62 von 90 Kategorien geschlagen. In 20 Kategorien war die SVM gleich auf mit dem k-Nearest-Neighbour Algorithmus.

Zusätzlich waren die SVM-Klassifizier unter allen getesteten Klassifizierern die performantesten. In der Trainings-Zeit ist die SVM mit dem C4.5 Algorithmus vergleichbar. Der Naive Bayes, Rocchio und der k-Nearest-Neighbour Algorithmus hatten die kürzesten Trainings-Zeiten.

Ähnliche Ergebnisse wurden auch bei einem zweiten Test mit den Oshumed-Datensatz [2][4] erzielt. Dabei wurde der von William Hersh kompilierte Trainingssatz genutzt, der 50216 Dokumente enthält. Davon wurden 10000 als Trainings-Daten und weitere 10000 als Test-Daten genutzt. Die Ergebnisse sind in Tabelle III dargestellt. Auch hier ist der k-Nearest-Neighbour der beste Klassifizerungsalgorithmus unter den konventionellen Algorithmen. C4.5 schneidet in diesem Test am schlechtesten ab. Wurden mehr als 500 Feature-Vektoren verwendet, konnte beim C4.5 sehr starke Überanpassung beobachtet werden. Die SVM schneidet besser ab als alle konventionellen Klassifizierer und schlägt den k-Nearest-Neighbour Klassifizierer in allen 23 Kategorien.

Tabelle III. BREAKEVENT-POINT DER 5 AM HÄUFIGSTEN VORKOMMENDEN OSHUMED-KATEGORIEN, SOWIE DER MIKRO-DURSCHSCHNITT. [2]

Kategorie	Bayes	Rocchio	C4.5	k-NN	SVM
Pathology	52,7	50,8	47,6	53,4	58,2
Cardiovascular	72,4	70,1	70,5	72,6	77,3
Immunologic	61,7	58,0	58,8	66,8	73,2
Neoplasms	63,6	64,1	58,7	67,2	70,6
Digestive System	65,3	59,9	59,0	67,1	73,7
Mikro-Durchschnitt	57,0	56,6	50,0	59,1	65,9

Beide Tests zeigen, dass die Support Vectore Machines eine starke Verbesserung gegenüber den konventionellen Klassifiziern darstellen. Die SVM hat in allen Konfigurationen und fast in allen Kategorien besser abgeschnitten als die konventionellen Klassifizier.

VI. FAZIT

Wie in den Kapiteln III und IV zu erkennen ist, haben die SVM's gegenüber den konventionellen, linearen Klassifizierern einige strukturelle Vorteile. Das Problem des Overfitting wird mit Hilfe ihrer zugrunde liegenden „Structural Risk Minimization“ weitestegehend gelöst. Zudem müssen nicht alle klassifizierten Dokumente, also Feature-Vektoren, zur Klassifizierung verwendet werden - es reicht aus, nur die Stützvektoren zu betrachten. Die hieraus resultierende Hyperebene führt zu einem enormen Performancevorsprung gegenüber den konventionellen Klassifizierern, da diese weitaus mehr Berechnungen bei der Klassifizierung durchführen müssen.

In der Praxis konnten sich die SVM's ebenfalls gegen die konventionellen Klassifizierer durchsetzenV-BV-A. Wie in den Kapiteln V-A und V-B zu erkennen ist, liegen die SVM's in Performance und Fehlerrate verglichen mit den konventionellen Klassifizierern sehr weit vorn. Zudem bieten die SVM's über die sogenannten Kerne noch Optimierungsmöglichkeiten, wie zum Beispiel in Kapitel V-A erwähnt, mit Invariantentransformationen, welche zusätzliche Optimierungen bei der Performance und der Fehlerrate ermöglichen.

An diesem Punkt könnten weitere Arbeiten angesetzt werden, um die Fragestellung nach weiteren Optimierungsmöglichkeiten und den damit verbundenen Performancevorteilen zu beantworten. Zudem ist der Einsatz von SVM's nicht auf linear separierbare Klassen oder die Textklassifizierung beschränkt, was ein breites Feld an Möglichkeiten eröffnet.

LITERATUR

[1] [Online]. Available: http://upload.wikimedia.org/wikipedia/de/a/a0/Diskriminanzfunktion.png

[2] T. Joachims, "A probabilistic analysis of the rocchio algorithm with tfidf for text categorization," in *Proceedings of the Fourteenth International Conference on Machine Learning*, ser. ICML '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 143–151.

[3] ——, "Text categorization with suport vector machines: Learning with many relevant features," *Proceedings of the 10th European Conference on Machine Learning*, pp. 137–142, 1998.

[4] [Online]. Available: https://www.mat.unical.it/OlexSuite/Datasets/SampleDataSets-download.htm

[5] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, Mar. 2002.

[6] [Online]. Available: http://de.wikipedia.org/wiki/Datei:Svm_intro.svg

[7] B. Schölkopf, K.-R. Müller, and A. J. Smola, "Lernen mit kernen: Support-vektor-methoden zur analyse hochdimensionaler daten," *Informatik Forschung und Entwicklung*, vol. 14, no. 3, pp. 154–163, Sep. 1999.

[8] D. Decoste and B. Schölkopf, "Training invariant support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 161–190, 2002.

[9] Simard, LeCun, and Denker, "Efficient pattern recognition using a new transformation distance," in *Advances in Neural Information Processing Systems 5*. San Mateo, CA: Morgan Kaufmann: Springer-Verlag, 1993, pp. 80–58.

[10] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Backpropagation applied to handwritten zip code recognition," in *Neural Computation 1*, 1989, pp. 541–551.

[11] G. Rätsch, B. Schölkopf, A. Smola, K.-R. Müller, T. Onoda, and S. Mika, "-arc: Ensemble learning in the presence of outliers," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 12*. MIT Press, 1999.

[12] M. Hearst, B. Schölkopf, S. Dumais, E. Osuna, and J. Platt, "Trends and controversies – support vector machines," *IEEE Intelligent Systems 13*, pp. 705–719, 1993.

[13] B. Schölkopf, "Support vector learning. doktorarbeit," München, 1997.

[14] V. Blanz, B. Scholkopf, H. Bülthoff, C. Burges, V. Vapnik, and T. Vetter, "Comparison of view-based object recognition algorithms using realistic 3d models," *Lecture Notes in Computer Science, Vol. 1112*, pp. 251–256, 1996.

[15] A. Smola, "Learning with kernels. doktorarbeit," *GMD Research Series No. 25*, 1998.

[16] K. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik, "Predicting time series with support vector machines," *Advances in Kernel Methods – Support Vector Learning*, pp. 243–253, 1999.

[17] H. Drucker, R. Schapire, and P. Simard, "Boosting performance in neural networks," *International Journal of Pattern Recognition and Artificial Intelligence 7*, pp. 18–28, 1998.

Erkennung und Synchronisation von Multimedia-Dateien

Emil Gabert
Hochschule Furtwangen
Fakultät Informatik
78120 Furtwangen, Deutschland
emil.gabert@hs-furtwangen.de

***Zusammenfassung*—In der heutigen Zeit gewinnen Multimedia-Anwendungen immer mehr an Bedeutung. Seit der Digitalisierung, beinhalten Musikdateien nicht mehr nur das reine Musikstück, sondern zusätzlich Meta-Daten. Diese Meta-Daten enthalten weitergehende Informationen, wie z.B. den Interpreten oder das Album, auf dem das Musikstück erschienen ist. Die Erkennung und Synchronisierung dieser Meta-Daten spielt in heutigen Multimedia-Anwendungen eine zentrale Rolle und ist Hauptteil dieser Ausarbeitung.**
Die Erkennung umfasst dabei das Einbinden und das Erkennen von USB-Datenträgern in einem Linux System. Die Synchronisation umfasst das Auslesen und das Speichern der Meta-Daten in einer Datenbank. Die Performanz dieses Prozesses spielt gerade im Embedded Bereich eine wichtige Rolle. Die Arbeit umfasst zusätzlich zu der Beschreibung des Synchronisationsprozesses eine Anbindung an einen Media-Player, der die synchronisierten Meta-Daten nutzt.
Im Embedded Bereich existieren häufig nur stark reduzierte Ressourcen. Das Projekt wurde deshalb auf einem Raspberry Pi 2 in Betrieb genommen und getestet. Für die Inbetriebnahme wurde ein auf das Projekt angepasstes Linux System erstellt

I. Einführung

In der Vergangenheit wurden das Erkennen und die Synchronisation von Multimedia-Dateien bereits untersucht, allerdings kam es hier zu keiner vollständigen Lösung. [1] Ziel dieser Arbeit war, die Erkennung und Synchronisation als eigenständige Module zu entwickeln und für weiterführende Arbeiten bereitzustellen. Die Erkennung umfasst dabei das Erkennen und Einhängen externer Datenträger, die Synchronisation beinhaltet das Auslesen und Speichern der in Musikstücken enthaltenen Meta-Daten. Die fertigen Module konnten in einem auf Node-Webkit basierten Player eingebunden und getestet werden. Dieser Multimedia-Player war Teil einer früheren Arbeit. [2]
Der auf Node-Webkit basierende Multimedia-Player dieser Arbeit soll konkurrenzfähig zu modernen Multimedia Systemen im Automotive Bereich sein. Dazu wurde die Funktionalität auf einem Embedded Gerät, dem Raspberry Pi 2 getestet, um die Gegebenheiten eines solchen Systems zu simulieren. Die Erkennung und Synchronisation von Multimedia-Dateien kann dabei als ein Teil des gesamten Systems betrachtet werden. Moderne Multimedia-Systeme im Automotive Bereich unterstützen häufig zusätzliche Funktionalität, die weit über die eines Multimedia-Players hinausgehen.

II. Erkennung

Dieser Abschnitt soll die grundsätzliche Erkennung von Geräten, im speziellen die Erkennung von USB-Geräten in Linux, erklären. Dabei war es notwendig, Geräteinformationen über USB-Datenträger während des laufenden Betriebs auszulesen. Es sollen vorhandene Konzepte und die in der Arbeit implementierte Lösung erläutert werden.

A. Dynamische Geräteerkennung

Verantwortlich für die Geräteerkennung in Linux Systemen ist der Kernel. Er kann als direkte Abstraktionsschicht zu der physikalischen Hardware verstanden werden. Ab der Kernel Version 2.6 findet die Geräteerkennung dynamisch mit Hilfe von udev statt.[1] Der Kernel legt erforderliche Geräteinformationen im sysfs-Dateisystem ab. Dabei wird für jedes Gerät das der Kernel erkennt, ein Ordner mit dem entsprechenden Gerätenamen erstellt. In diesem Ordner befinden sich dann gerätespezifische Attributdateien. [3]

B. udev

Für die dynamische Geräteverwaltung, ist udev zuständig. Eine Änderung des Gerätestatus muss vom Kernel an den Userspace weitergegeben werden. Dazu verwendet der Kernel uevents. Die udev Infrastruktur beinhaltet den udev-Daemon, der auf die gesendeten uevents hört und die Einrichtung und Konfiguration von Geräten im Userspace initiiert. Udev legt dann die Gerätedateien im /dev Verzeichnis dynamisch an oder entfernt diese. Somit spiegelt das /dev Verzeichnis immer den aktuellen Zustand des Kernels wider.

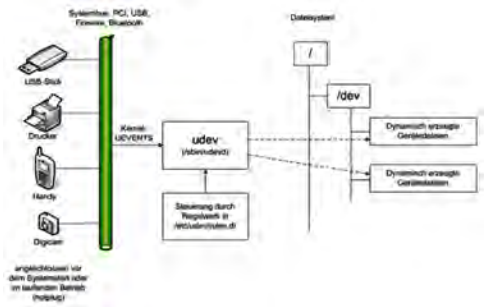


Abbildung 1: Ablauf der Erkennung.[2]

[1] https://de.wikipedia.org/wiki/Udev

Der gesamte Prozess der Erkennung funktioniert während des Bootvorgangs und während des laufenden Betriebs. Während des Bootvorgans ist der udev-Daemon noch nicht gestartet. Hier behilft sich der Kernel mit einem einfachen Trick. Der Kernel legt die Ereignisse in einer uevent Datei ab, die sich im sysfs-Dateisystem befindet. Sobald der udev-Daemon gestartet ist fordert er den Kernel auf, die verloren gegangenen Ereignisse während des Bootvorgans erneut zu senden. Ab diesem Zeitpunkt gibt es keinen Unterschied zur Erkennung von Hardware im laufenden Betrieb. [4]

C. Implementierung

Für die Implementierung des Moduls zur Erkennung von Datenträgern, wurde die libudev verwendet. In dieser Arbeit war es notwendig, über Änderungen des Gerätestatus informiert zu werden. Dazu bietet libudev einen Monitor, der es ermöglicht, über udev Ereignisse informiert zu werden. Auf diesen Monitor lässt sich ein Filter setzen, um nur über Änderungen bestimmter Geräte informiert zu werden. Die Ereignisse lassen sich wie folgt kategorisieren:

- add - Ein Gerät wurde eingesteckt
- remove - Ein Gerät wurde ausgesteckt
- change - Ein Gerät hat sich verändert
- move - Geräteknoten wurde verschoben oder umbenannt

In der Arbeit wurde ein Monitor für Blockgeräte erstellt, um über Ereignisse für USB-Datenträger informiert zu werden. Zusätzlich wurde auf das Erkennen von Partitionen gefiltert.

```
udev_monitor_filter_add_match_subsystem_devtype(mon,
    "block", "partition");
```

Sobald ein Ereignis der Kategorie add erkannt wird, wird zusätzlich der Einhängepunkt im System ermittelt. Eine Auflistung aller aktuell eingehängten Datenträger in einem Linux-System, findet sich unter /proc/mounts. Durch den erkannten Geräteknoten ist es möglich, den Einhängepunkt durch das Auslesen der Datei zu ermitteln. Linux bietet dazu die Headerdatei mntent.h, die diese Funktionalität bereitstellt.

Für das automatische Einhängen von Datenträgern wird das Programm usbmount verwendet. Usbmount lässt sich mit der Datei /etc/usbmount/usbmount.conf frei konfigurieren. So können z.B. die Rechte oder bestimmte Mount-Parameter gesetzt werden.

Die Erkennung von Datenträgern ist in dem Modul Mount.cpp implementiert. Das Modul stellt folgende öffentliche Methoden bereit

```
void startScan(string &mountpoint, string &deviceID,
    string &deviceNode, bool &attached)
```

Die Methode initialisiert den Monitor und wartet anschließend in einer Schleife auf udev Ereignisse. Die Methode ist blockierend und gibt folgende Informationen zurück sobald ein Gerät erkannt wurde.

- mountpoint - Einhängepunkt im Dateisystem
- deviceID - ID bestehend aus Hersteller und Produkt ID
- deviceNode - Geräteknoten im /dev Verzeichnis
- attached - Kategorie add oder remove

```
void getMountpoint(string &deviceNode, string &mountpoint)
```

Die Methode ermittelt den Einhängepunkt zu einem gegebenen Geräteknoten. Wurde das Gerät nicht automatisch eingehängt, liefert die Methode einen leeren String zurück. Dabei ist der Benutzer für das automatische Einhängen des Datenträgers verantwortlich. Sobald ein Gerät erkannt wurde, ruft die Methode startScan die Methode getMountpoint auf, um den aktuellen Einhängepunkt zu ermitteln.

D. Optimierung

Bei der Implementierung des Moduls zur Erkennung ist aufgefallen, dass es sehr lange dauert, USB-Datenträger zu erkennen. Beim Einstecken eines USB-Gerätes muss der Kernel Informationen über das erkannte Gerät erhalten. Dazu befindet sich auf dem USB Gerät ein Microcontroller, der unter anderem dazu verwendet wird, ein USB Gerät zu identifizieren. Das Auslesen der Informationen übernimmt der Kernel und dies geschieht im Wesentlichen in vier Schritten:[3]

1) Erkennung der Gerätegeschwindigkeit
   - low speed 1.5 Mbps
   - full speed 12 Mbps
   - high speed 480 Mbps
2) Erkennung des Gerätetyps (Device Descriptor)
   - Host initiiert ein Reset.
   - Host sendet Get Device Descriptor.
   - Host empfängt Deskriptoren z.B. Vendor ID, Product ID
3) Erkennung der Gerätekonfiguration (Configuration Descriptor)
   - Feste Größe 9 Bytes
   - Informationen wie z.B. wieviele Geräte unterstützt werden und die maximale Stromaufnahme
4) Erkennung des Geräte Interfaces (Interface Descriptor)
   - Feste Größe 9 Bytes
   - Informationen wie z.B. die Anzahl der Endpunkte (in and out)

Der Erkennungsprozess kann nicht optimiert werden, da nicht in die Gerätesoftware eingegriffen werden kann. Es gibt aber einen Kernel Parameter, (usb.storage.delay.use) der speziell für USB-Datenträger eingeführt wurde. Das Delay existiert, um Datenträgern zusätzlich Zeit für die Initialisierung zu geben. Vor 2005 war die Standardeinstellung dieses Parameters noch auf fünf Sekunden gesetzt. Nach 2005 wurde das Delay von Linus Torvalds auf eine Sekunde reduziert. Für das Projekt wurde der Kernel-Parameter auf null Sekunden heruntergesetzt. [5]

[2] Quelle: http://www.linux-community.de

[3] http://tinyurl.com/k8vvoty

Das Anpassen des Kernelparameters ist nicht in das Modul integriert und muss bei der Inbetriebnahme manuell gesetzt werden. Viele moderne Linux Distributionen haben den Standardwert dieses Delays aber ohnehin bereits auf null Sekunden reduziert. [5]

Um die Änderung während des laufenden Betriebs zu testen, kann folgendes Kommando ausgeführt werden:

```
echo 0 > /sys/module/usb_storage/parameters/delay_use
```

Um den Parameter permanent zu ändern, muss er dem Kernel beim Start übergeben werden. Die Konfiguration ist dabei abhängig vom verwendeten System.

```
usb-storage.delay_use=<Zeit in Sekunden>
```

III. Synchronisation

Die Synchronisation beinhaltet das Auslesen der Meta-Daten aus Musikstücken und die Speicherung dieser Meta-Daten. Im Projekt wurde für die Speicherung der Daten eine SQLite[4] Datenbank eingesetzt.

A. ID3-Tags

ID3-Tags werden genutzt, um Meta-Daten in mp3-Dateien in einem einheitlichen Format zu speichern. Der Vorteil von ID3-Tags ist, dass das Musikstück die Meta-Daten selbst speichert. Vor der Einführung von ID3-Tags mussten diese Informationen aus dem Dateinamen ausgelesen werden, was zu mehreren Nachteilen führte. Die Dateinamen wurden lang und unübersichtlich, des Weiteren unterstützen viele Dateisysteme nicht alle Sonderzeichen. ID3-Tags sind nicht Bestandteil des MP3-Standards.[5] Im Projekt wurde für das Auslesen und Parsen die TagLib verwendet.[6]

B. Optimierung

Das Auslesen der Meta-Daten nimmt enorm viel Zeit in Anspruch. Das ist unter Anderem auf das Auslesen des Album Covers zurückzuführen. Das Album Cover ist eine Bilddatei. Diese Bilddatei muss im JPEG oder PNG Format vorliegen und darf laut Spezifikation des ID3v2.4 Formats bis zu 256MB groß sein. [6] Häufig sind moderne Endgeräte hochauflösend, weshalb immer mehr Implementierungen hochauflösende Bilder in den Tags speichern. Die Größe des Tags wächst dadurch erheblich und die Auslesegeschwindigkeit reduziert sich dadurch meist auf die Lesegeschwindigkeit des Datenträgers. Gerade bei USB 2.0 Datenträgern macht sich das negativ bemerkbar.

Ein möglicher Ansatz, die Auslesegeschwindigkeit zu erhöhen, besteht darin, das Album Cover nur bei Bedarf auszulesen und nicht direkt bei der Synchronisation.

[4]https://www.sqlite.org
[5]https://de.wikipedia.org/wiki/ID3-Tag
[6]https://taglib.github.io

C. Datenbank

Für die Persistierung der ausgelesenen Meta-Daten, wurde sich im Rahmen des Projektes für eine SQLite Datenbank entschieden. Durch die Datenbank soll dem Benutzer die Möglichkeit gegeben werden, Musikstücke einfach und kategorisiert auswählen zu können. Das Durchsuchen eines Datenträgers beschränkt sich durch die Datenbank nicht mehr nur auf reines Datei-Browsing, sondern bietet die Möglichkeit, Musikstücke nach dem Genre oder dem Album auf dem das Musikstück erschien, auszuwählen.

D. Aufbau

Die Datenbank soll die ausgelesenen Meta-Daten möglichst effizient speichern. Eine Besonderheit der im Projekt verwendeten Datenbank ist, dass zusätzlich zu den Meta-Daten Datenträger gespeichert werden können. Durch das Speichern der Datenträger ist es möglich, bereits synchronisierte Datenträger wiederzuerkennen, um den Synchronisationsprozess zu beschleunigen. Die Musikstücke speichern zusätzlich Dateiattribute. Durch die Speicherung der Attribute ist es möglich, bereits synchronisierte Musikdateien wiederzuerkennen und im Synchronisationsprozess zu überspringen. Durch diesen Aufbau der Datenbank wird erreicht, dass der Synchronisationsprozess lediglich beim Erkennen einer neuen Datei die Meta-Daten ausliest.

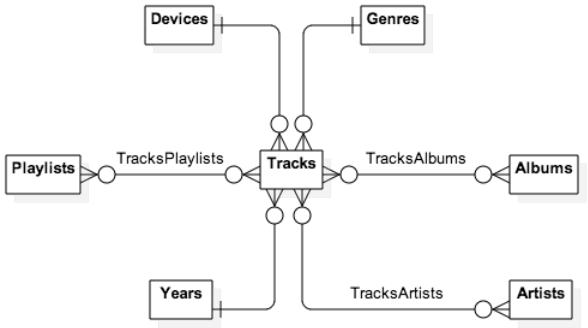


Abbildung 2: ER-Modell SQLite Datenbank

Abbildung 2 zeigt das ER-Modell der im Projekt verwendeten Datenbank. Die Tabelle Devices speichert Informationen über bereits synchronisierte Datenträger. Die Datenträger werden durch die Vendor und Product ID identifiziert. Die Tabelle Tracks speichert Informationen über bereits synchronisierte Musikstücke. Die Musikstücke enthalten zusätzlich zu den Meta-Daten Dateiattribute. Durch die Dateiattribute ist es möglich, bereits synchronisierte Dateien wiederzuerkennen. Im Projekt wird der C++ Wrapper SQLiteCpp benutzt.[7] Der Wrapper stellt wenige gut dokumentierte Klassen für den Zugriff auf die SQLite C API zur Verfügung. Die Methoden für die Erstellung der Datenbank und den Zugriff auf die Datenbank sind in der Klasse SongDatabase.cpp implementiert. Sie bietet die öffentliche Methode.

```
void synchDatabase(const std::string &directory, const
    std::string &device_id);
```

[7]https://github.com/SRombauts/SQLiteCpp

Die Methode bekommt das zu synchronisierende Verzeichnis und die Device ID übergeben. Die Device ID kann früher aus der Methode für die Erkennung startScan ermittelt werden. Bei der Ausführung der Methode wird das angegebene Verzeichnis rekursiv durchlaufen und alle gesammelten Informationen in der Datenbank gespeichert.

Die Erstellung der Datenbank realisiert der Konstruktor, der lediglich den Dateinamen übergeben bekommt.

```
SongDatabase(std::string filename);
```

Der Konstruktor überprüft, ob die angegebene Datei vorhanden ist. Falls nicht, wird die Datenbank mit der privaten Methode createDatabase erstellt.

Das Hinzufügen, Entfernen und Aktualisieren einzelner Datenbankeinträge übernehmen mehrere private Methoden. Sie folgen einem ähnlichen Aufbau.

```
int SongDatabase::insertArtist(const Artist &artist)
{
  //Neues Statement erzeugen falls noetig
  if (!stmtInsArtist)
  {
    stmtInsArtist =
        unique_ptr<Statement>(
           new Statement(*db,
              "INSERT OR IGNORE INTO Artists (ArtistName)
                  VALUES(?)"));
  }
  int result = 0;
  //Binden des Kuenstlernamens an ersten Parameter des
       Statements
  stmtInsArtist->bind(1, artist.artistName);
  //Statement ausfuehren
  result = stmtInsArtist->exec();
  //Statement zuruecksetzen
  stmtInsArtist->clearBindings();
  stmtInsArtist->reset();
  return result;
}
```

Als Beispiel soll hier das Einfügen eines neuen Künstlers beschrieben werden. Der Künstler und alle anderen Tabellen sind in entsprechenden Klassen gekapselt, hier die Klasse Artist. Die Methode erzeugt bei Bedarf ein neues SQL Statement. Zu beachten ist, dass der Künstler nur eingefügt wird, wenn er nicht bereits vorhanden ist. Dafür sorgt INSERT OR IGNORE. Nach der Ausführung des Statements mit stmtInsArtist->exec() steht in dem Feld result die eingefügte ID. Diese ID wird am Ende der Methode an den Aufrufer zurückgegeben.

Die verschiedenen Statements, zu erkennen an dem Prefix stmt, sind Klassenvariable und als Smart-Pointer implementiert worden. Smart-Pointer löschen beim Verlassen ihres Scopes selbständig das erzeugte Objekt auf dem Heap. So ist es nicht notwendig, den Speicher manuell mit delete freizugeben. Um Smart-Pointer benutzen zu können, muss lediglich die memory Header-Datei eingebunden werden. [7]

E. Synchronisationsprozess

Die Datenbank speichert nicht nur die eigentlichen Meta-Daten des Musikstücks, sondern zusätzlich Informationen über den synchronisierten Datenträger und Attribute der synchronisierten Dateien. Durch diese Informationen wird es ermöglicht, bereits synchronisierte Dateien wiederzuerkennen und das Auslesen der Meta-Daten zu verhindern. Es wurde bereits darauf eingegangen, dass das Auslesen der Meta-Daten viel Zeit kostet und stark von der Lesegeschwindigkeit des Datenträgers abhängig ist.

Im Projekt wurde die Optimierung des Synchronisationsprozesses mit Hilfe eines Hashvergleichs der zu synchronisierenden Dateien mit der Datenbank gelöst. Dabei werden der Pfad, die Größe und das Erstellungsdatum und das Änderungsdatum benutzt. Aus diesen Dateiattributen wird ein Hash gebildet und mit den bereits synchronisierten Musikdateien in der Datenbank verglichen. Abbildung 3 zeigt einen vereinfachten Ablauf des Synchronisationsprozesses. Zunächst werden alle Attribute verglichen. Finden sich die

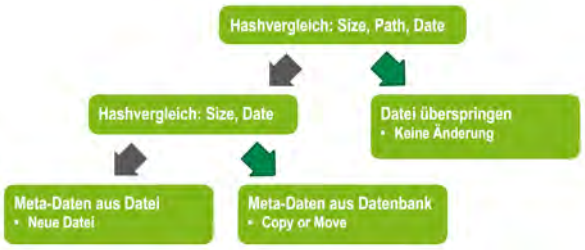


Abbildung 3: Ablauf Synchronisationsprozess

Attribute in der Datenbank, kann die Datei übersprungen werden und das Auslesen der Meta-Daten findet nicht statt. Des Weiteren kann es vorkommen, dass der Benutzer zwischen zwei Synchronisationsabläufen Dateien verschiebt oder umbenennt. In diesem Fall müssen die Meta-Daten des Musikstücks wieder nicht ausgelesen werden und lediglich der Pfad der Datei in der Datenbank geändert werden. Falls die Größe, das Erstellungsdatum oder das Änderungsdatum der Datei nicht in der Datenbank vorhanden ist, müssen die Meta-Daten der Datei ausgelesen und in der Datenbank gespeichert werden. In diesem Fall handelt es sich um eine neue oder geänderte Datei.

Der Synchronisationsalgorithmus wurde getestet, indem 1000 Musikdateien von einem externen Datenträger synchronisiert wurden. Die Dateien wurden während des ersten und zweiten Durchlaufs des Algorithmus nicht geändert. Durch

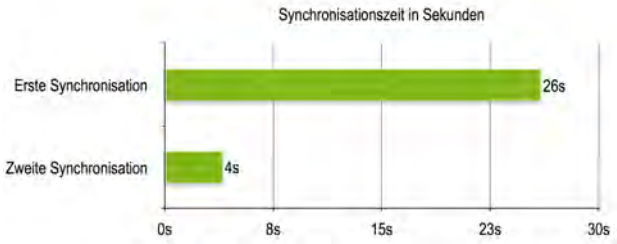


Abbildung 4: Test Synchronisationsalgorithmus

das Überspringen bereits synchronisierter Dateien wurde die Synchronisationszeit beim zweiten Durchlauf um 22

Sekunden reduziert. Die Zeit für die Synchronisation reduziert sich im zweiten Durchlauf im Wesentlichen auf die Datenbankabfragen und das Auslesen der Dateiattribute. Da sich die Dateien nicht verändert haben, müssen die Meta-Daten nicht ausgelesen werden.

IV. Entwicklung eines nativen Players

A. GStreamer

Das Abspielen von Musikdateien wird mit Hilfe der Gstreamer Engine realisiert.[8] Die Gstreamer Engine ist unter Linux ein weit verbreitetes Framework zum Erstellen von Streaming Media Anwendungen. Gstreamer ist pluginbasiert und schafft dadurch eine Umgebung, bei der die Implementierung eines Players nicht auf Details, wie den benötigten Codec, oder gar die Hardware achten muss. Durch diesen Ansatz ist es möglich, einen auf GStreamer basierenden Player zu entwickeln und automatisch von Neuerungen und Änderungen an der Engine zu profitieren. [8]

B. Implementierung

Im Rahmen der Arbeit sollte für das Abspielen von Musikdateien ein nativer, auf Gstreamer basierender Media-Player entwickelt werden. Der Media-Player sollte in C++ entwickelt werden und folgende Features bieten:

- Musikdateien öffnen - gängige Formate mp3, ogg
- Musikdateien abspielen
- Musikdateien pausieren
- Musikdateien stoppen
- Musikdateien spulen

Der Media-Player ist im Projekt in der Datei GstPlayer.cpp implementiert. Es stehen die folgenden öffentlichen Methoden bereit.

```
bool open(const std::string &uri)
```

Die Methode bekommt die abzuspielende Musikdatei übergeben und initialisiert die Gstreamer Engine. Wurde die Datei erfolgreich geöffnet, gibt die Methode true zurück, ansonsten false.

```
void play();
```

Die Mehode spielt die momentan geöffnete Datei ab.

```
void pause();
```

Falls eine Musikdatei gerade abgespielt wird, wird sie pausiert. Mit play() kann das Abspielen der Musikdatei wieder fortgesetzt werden.

```
void seek(long position);
```

Die Methode bekommt die Position innerhalb des Musikstücks in Millisekunden übergeben. Für die Benutzung der Methode stehen zwei weitere nützliche Methoden wie getPosition und getDuration bereit.

```
State getState();
```

[8] http://gstreamer.freedesktop.org

Die Methode gibt den aktuellen Status des Players zurück. Mögliche Stati sind dabei:

- NONE - Gstreamer wurde nicht mit einer Musikdatei initialisiert.
- READY - Gstreamer wurde initialisiert, aber spielt die Musikdatei noch nicht ab.
- PAUSED - Das Abspielen wurde pausiert.
- PLAYING - Die Musikdatei wird abgespielt.

Zusätzlich zu den oben genannten Methoden existiert ein Event basierter Ansatz zur Information über den Wechsel der verschiedenen Stati. Um das Event zu abonnieren, steht im GstPlayer folgende Methode zur Verfügung.

```
sigc::signal<void, State>& signalStateChanged();
```

Die Methode gibt eine Referenz auf das Signal zurück. Durch das einfache Verbinden einer Methode mit dem Signal wird der Benutzer über Statusänderungen des Players informiert. Die Benutzung der Methode gestaltet sich wie folgt.

```
GstPlayer player;
player.signalStateChanged().connect(sigc::ptr_fun(&stateChanged));
```

Mit Hilfe der Methode connect des Signals, kann eine eigene Methode in diesem Fall stateChanged angegeben werden, die aufgerufen wird, sobald sich der Status ändert.

```
void stateChanged(GstPlayer::State state)
{
      switch(state)
      {
      case GstPlayer::NONE:
            cout << " switched to NONE" << endl;
            break;
      case GstPlayer::PAUSED:
                  cout << " switched to PAUSED" << endl;
                  break;
      case GstPlayer::PLAYING:
                  cout << " switched to PLAYING" << endl;
                  break;
      case GstPlayer::READY:
                  cout << " switched to READY" << endl;
                  break;
      }
}
```

In diesem Fall wird lediglich auf der Console ausgegeben, in welchem Status sich der Player momentan befindet.

V. Entwicklung eines webbasierten Players

A. Node.js

Node.js ist eine serverseitige Plattform für Netzwerkanwendungen. Ursprünglich war Node.js dafür gedacht, Webserver zu entwickeln. Node.js verwendet Googles JavaScript-Engine V8, die auch im Google Chrome Browser zum Einsatz kommt. Node.js enthält zahlreiche Module, die in die Anwendung kompiliert wurden. Zusätzlich ist es möglich, eigene Module in C++ oder JavaScript zu schreiben, um den Funktionsumfang von Node.js zu erweitern. Dadurch wird erreicht, dass man nativen C++ Code schreiben

kann, der später durch JavaScript aufgerufen wird. [9]

Zur Verwaltung dieser Module bietet Node.js den Paketmanager NPM an. Während der Erstellung der Arbeit, wurde ein neues Framework - das IO.js - released. Die erste Version von IO.js wurde am 13. Januar 2015 released und ist lediglich ein Fork von Node.js. Die Unterschiede sind bisher marginal, so unterstützt IO.js den NPM Paketmanager und ist voll kompatibel zu Node.js.[9]

B. NWjs

NWjs, früher Node-Webkit, wurde im Intel Open Source Technology Center entwickelt. NWjs basiert auf dem Chromium Open Source Browser und enthält zusätzlich Node.js. Dadurch ist es möglich, direkt aus dem Document Object Model Node.js Module aufzurufen. Es ist also möglich, Anwendungen in modernem HTML5, CSS3, JavaScript und zusätzlich WebGL zu schreiben. Zusätzlich zu den Webtechnologien unterstützt NWjs alle Node.js APIs sowie alle selbst erstellten Node.js Module. Dadurch wird es möglich, eigene Module in reinem C++ zu schreiben und diese dann als Node.js Module zu bauen und aus dem Browser heraus aufzurufen. [10]

C. Implementierung

Zur Entwicklung des Players für NW.js konnte ein Player aus einer früheren Arbeit herangezogen werden. [2] Ziel war es, die in reinem C++ geschriebenen Module zunächst für Node.js/IO.js zur Verfügung zu stellen. Zu diesem Zweck wurde ein Controller geschrieben. Der Controller besteht aus zwei Klassen, dem DatabaseWorker und dem PlayerWorker.

```
class DatabaseWorker: public NanAsyncWorker

class PlayerWorker: public NanAsyncProgressWorker
```

Für die Synchronisation und den Player wurde jeweils ein asynchroner Ansatz benötigt. Der Benutzer wird über einen Callback informiert, sobald die Synchronisation abgeschlossen wurde. Die PlayerWorker Klasse arbeitet nach einem ähnlichen Prinzip. Hier ist es nicht nur möglich, bei Beendigung des Players, sondern auch während der Ausführung über Statusänderungen informiert zu werden. Der Player ist in der Ausführung über den gesamten Ablauf des Programms blockierend. Die beiden Klassen erben von NanAsyncWorker. Um den asynchronen Ablauf zu gewährleisten müssen zwei Methoden überschrieben werden. Der Vorgang soll hier beispielhaft für die Synchronisation beschrieben werden.

```
void Execute()
{
      m.startScan(mountpoint, device_id, device_node,
          attached);
      if (attached)
      {
            std::cout << "Start synching" << std::endl;
            db.synchDatabase(mountpoint, device_id);
      }
}
```

In dieser Methode findet die asynchrone Ausführung statt. Bei der Synchronisation wird hier asynchron auf das Einstecken eines Datenträgers gewartet und sobald ein Gerät erkannt wurde, die Synchronisation gestartet. Für den Callback ist die Methode HandleOkCallback zuständig, die aufgerufen wird, sobald die asynchrone Ausführung aus Execute endet.

```
void HandleOKCallback()
{
      NanScope();
      Local<Value> argv[] =
      {
            NanNew<v8::String>(mountpoint),
            NanNew<v8::String>(device_id),
            NanNew<v8::String>(device_node),
            NanNew<v8::Boolean>(attached)
      };
      callback->Call(4, argv);
}
```

An dieser Stelle werden die Rückgabeparameter zunächst vorbereitet und im Anschluss ein Callback mit den Parametern aufgerufen. Hier wird auch die Schnittstelle von C++ zu Javascript deutlich. Die Rückgabe wird für die V8 Engine aufbereitet, bevor der Callback stattfindet. Der Benutzer ruft den DatabaseWorker später aus Javascript heraus auf und bekommt die Rückgabeparameter in gewohnter Form übergeben.

Für die Benutzung des DatabaseWorkers steht im Controller die Methode Mount zur Verfügung.

```
NAN_METHOD(Mount)
{
      NanScope();
      NanCallback *callback = new
           NanCallback(args[0].As<Function>());
      NanAsyncQueueWorker(new DatabaseWorker(callback));
      NanReturnUndefined();
}
```

Die Methode instanziiert den DatabaseWorker mit der übergebenen Javascript Callback Methode. Auf der Javascript Seite gestaltet sich der Aufruf dann sehr einfach.

```
player=require('./build/Release/NodePlayer.node');
player.Mount(onMount);
```

Zunächst wird das kompilierte Node Modul eingebunden und anschließend kann die Mount Methode genutzt werden. Die C++ Seite trägt Sorge dafür, dass onMount mit den vom Modul zurückgegebenen Parametern aufgerufen wird.

```
function onMount(mountpoint, device_id, device_node,
     attached)
{
      if(attached)
      {
            addDevice(mountpoint, device_node, device_id);
      }
      else
      {
            removeDevice(device_node);
      }
      player.Mount(onMount);
}
```

Die Methode fügt ein erkanntes Device, abhängig vom attached Parameter, hinzu oder entfernt dieses und ruft im Anschluss wieder die Mount Methode auf. Diese Implementierung hat den Nachteil, dass neue Geräte während eines Synchronisationsprozesses nicht erkannt und somit

[9] https://de.wikipedia.org/wiki/Node.js

auch nicht synchronisert werden. Hier könnte ein anderer Ansatz in einer weiterführenden Arbeit gewählt werden. Das Synchronisieren von mehreren Geräten gleichzeitig könnte z.B. mit einer Warteschlange realisiert werden.

Nach dem Synchronisationsprozess beinhaltet die Datenbank alle Informationen über die synchronisierten Musikstücke. Für das Abrufen und Anzeigen der Daten aus der Datenbank wurde das in NPM zur Verfügung stehende sqlite3 Modul verwendet. Um auf die synchronisierte Datenbank zugreifen zu können, muss zunächst das Node.js Modul eingebunden werden.

```
var sqlite = require('sqlite3');
```

Der Zugriff auf die Datenbank findet im Projekt in der Methode getSongs statt.

```
function getSongs(deviceName, callback)
{
      var tempSongs = new Array;
      var db = new sqlite.Database('/home/pi/node.db');
      var selectStr = "SELECT ArtistName, AlbumName,
           GenreName, TrackTitle, TrackURI FROM Artists,
           Albums, Genres, Tracks WHERE
           Tracks.AlbumID=Albums.AlbumID AND
           Tracks.ArtistID=Artists.ArtistID AND
           Tracks.GenreID=Genres.GenreID AND
           Tracks.DeviceID=(SELECT DeviceID FROM Devices
           WHERE DeviceName='" + deviceName + "')";
      db.all(selectStr, function(err, rows)
      {
            rows.forEach(function (row)
            {
                  var song = new Song(row.ArtistName,
                       row.AlbumName, row.GenreName,
                       row.TrackTitle, row.TrackURI);
                  tempSongs.push(song);
            });
            callback(tempSongs);
      });
      db.close();
}
```

Es werden alle Musikstücke aus der Datenbank ausgelesen und im Anschluss ein Menü für die Anzeige erstellt, in dem der übergebene Callback mit den ausgelesenen Musikstücken aufgerufen wird. Die gesamte Funktionalität für den in Javascript geschriebenen Player findet sich im Projekt in der Datei player.js.

VI. Fazit

Die Zielsetzung für das Projekt konnte im Rahmen der Thesis erreicht werden. Die Module für die Erkennung und Synchronisierung von Meta-Daten wurden in C++ entwickelt und in einen auf NWjs basierenden Player eingebunden. Die Synchronisation umfasst dabei das Auselesen und die Persistierung von Meta-Daten. Zusätzlich konnte ein angepasstes Linux für den Raspberry Pi 2 erstellt werden. Damit ergeben sich folgende erreichte Ziele:

- Erkennung externer Speichermedien
- Auslesen von Meta-Daten
- Persistierung der ausgelesenen Meta-Daten
- Entwicklung eines nativen Media-Players
- Entwicklung eines webbasierten Media-Players

Es konnte gezeigt werden, dass die Umsetzung eines auf NWjs basierenden Media-Players auch auf langsamen Zielgeräten wie dem Raspberry Pi 2 möglich ist. Durch die Zusammenführung der Performanz von nativ programmierten Modulen und der Einfachheit moderner Webtechnologien, bietet diese Thesis einen guten Ausgangspunkt für kommende Arbeiten über Multimedia-Anwendungen für Embedded Systeme.

Danksagung

„In jede hohe Freude mischt sich eine Empfindung der Dankbarkeit.“ [11, S. 33]

Mein Dank gilt allen, die an der Durchführung dieser Thesis beteiligt waren. Namentlich möchte ich meinen Betreuern Herrn Prof. Dr. Lothar Piepmeyer und Herrn Prof. Dr. Elmar Cochlovius für die ausgezeichnete Unterstützung in allen Phasen des Projekts danken. Mein weiterer Dank gilt Herrn Ralf Schellhammer für die Unterstützung bei der Beschaffung der Hardware.

Literatur

[1] C. Trost, *Erkennung und Synchronisation von Multimedia-Dateien auf mobilen Plattformen*. Bachelorthesis, Hochschule Furtwangen, 2014.

[2] S. Held, *Konzeption und Implementierung einer platt- formunabhängigen, multimedialen Anwendung*. Bachelorthesis, Hochschule Furtwangen, 2014.

[3] G. Kroah-Hartman, “udev – A Userspace Implementation of devfs,” 2003. [Online]. Available: https://www.kernel.org/doc/ols/2003/ols2003-pages-249-257.pdf

[4] SUSE Linux Enterprise Server Verwaltungshandbuch. [Online]. Available: https://www.suse.com/de-de/documentation/sles11/singlehtml/book_sle_admin/cha.udev.html

[5] L. Torvalds. Lower usb storage settling delay to something more reasonable. [Online]. Available: http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=a4a47bc03fe520e95e0c4212bf97c86545fb14f9

[6] Id3 tag version 2.4.0 - main structure. [Online]. Available: http://id3.org/id3v2.4.0-structure

[7] B. Stroustrup, *A tour of C++*, B. Stroustrup, Ed. Upper Saddle River, NJ: Addison Wesley, c2014. [Online]. Available: http://proquest.tech.safaribooksonline.de/9780133549041

[8] W. T. et al. Gstreamer application development manual (1.5.2). [Online]. Available: http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/

[9] M. Wandschneider, *Learning Node.js: a hands-on guide to building Web applications in JavaScript*. Upper Saddle River, NJ: Addison Wesley, 2013. [Online]. Available: http://proquest.tech.safaribooksonline.de/9780133377996

[10] Nwjs. [Online]. Available: https://github.com/nwjs/nw.js

[11] M. v. Ebner-Eschenbach and K. Krolow, *Aphorismen* -, 25th ed. Frankfurt, Leipzig: Insel-Verlag, 1986.