Bauhaus-Universität Weimar
Fakultät Medien
Studiengang Mediensysteme

Entwicklung einer portablen Software zur Plagiatanalyse

Bachelorarbeit

Alexander Kümmel — Matrikelnummer 30175
geb. am: 22.01.1984 in St. Petersburg

1. Betreuer: Prof. Dr. Benno Stein
2. Betreuer: Martin Potthast

Datum der Abgabe: 27. Mai 2008

Bauhaus-Universität Weimar
Fakultät Medien
Studiengang Mediensysteme

Entwicklung einer portablen Software zur Plagiatanalyse

Bachelorarbeit

Alexander Kümmel Matrikelnummer 30175
geb. am: 22.01.1984 in St. Petersburg

1. Betreuer: Prof. Dr. Benno Stein
2. Betreuer: Martin Potthast

Datum der Abgabe: 27. Mai 2008

Erklärung

Hiermit versichere ich, dass ich diese Arbeit ohne fremde Hilfe und allein mit den angegebenen Quellen angefertigt habe und dass diese Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat.

Weimar, den 15. Juli 2008 Alexander Kümmel

Inhaltsverzeichnis

1 Motivation **1**

2 Softwarearchitektur und Entwicklung **4**
2.1 Paradigmen der Softwareentwicklung 4
2.1.1 Abstraktion 5
2.1.2 Softwarestrukturierung 5
2.1.3 Modularisierung 6
2.1.4 Softwarequalität 8
2.2 Softwarearchitekturen im Überblick 9
2.2.1 Monolithische Software 9
2.2.2 Client-Server-Software 10
2.2.3 Dienstorientierte Software 10
2.2.4 Web-Dienste 11
2.3 Komponentenorientiertes Softwaredesign 12
2.4 Entwurf einer portablen Software 21

3 Das Plagiatanalysesystem Picapica **26**
3.1 Prozessmodell der Plagiatanalyse 26
3.1.1 Kernaufgaben der Plagiatanalyse 27
3.1.2 heuristisches Retrieval 27
3.1.3 detailierte Analyse 27
3.1.4 Nachbearbeitung 29
3.2 Ausführungsmodell und Server-Architektur 29
3.2.1 Analyse 31
3.2.2 Kommunikation und Datenaustausch 31
3.2.3 Benutzerinteraktion 32
3.3 Web-basierte Visualisierung von Analyseergebnissen. 33
3.4 Softwarekomponenten 36
3.4.1 Interne Komponenten 36
3.4.2 Externe Komponenten 38
3.5 Portable Plagiatanalyse 39

4 Zusammenfassung **42**

Literaturverzeichnis **43**

1 Motivation

In Zeiten zunehmender, globaler Vernetzung erfreut sich das Internet großer Beliebtheit. Das World Wide Web, ein Teilbereich des Internet, bietet zahllose gut ausgearbeitete Inhalte zu unterschiedlichen Themenbereichen. Die Menge themenspezifischer Texte hat längst unüberschaubare Ausmaße erreicht und kann von einem Menschen nicht mehr überblickt werden. Aus diesem Grund kommt es beim Schreiben neuer Publikationen immer häufiger dazu, dass Autoren den einfachen Weg wählen und die Arbeiten anderer Autoren stückweise kopieren und in die eigenen Texte einflechten, anstatt sie ausreichend zu zitieren. Die Aufgabe der Plagiatanalyse besteht darin, diese Vorgehensweise einiger Autoren zu erkennen und Plagiate als solche zu identifizieren. Ein Plagiat versteht sich als die Vorlage fremden, geistigen Eigentums als etwas eigenes. Einem Menschen fällt es leicht, einen Textabschnitt als mögliches Plagiat zu erkennen. Die ursprüngliche Quelle des Plagiats wiederzufinden, ist angesichts der Menge an Publikationen nur noch maschinell möglich. Für das Auffinden der Quelle ist ein stückweiser Vergleich des plagiierten Textabschnitts mit allen anderen verfügbaren Texten nötig. Erst dieser direkte Vergleich der Textabschnitte, erlaubt bei einem übereinstimmenden Ergebnis eine hinreichende Aussage darüber, ob ein Plagiat vorliegt. Aufgrund der zahllosen Texte, die über das WWW verfügbar sind, ist diese Suche von einem Menschen nicht mehr zu bewältigen. Hier wird die Verarbeitungsleistung des Computers bzw. eines Softwaresystems notwendig. Die Software ist im Gegensatz zum Menschen in der Lage, große Dokumentkollektionen in kurzer Zeit zu verarbeiten und für einen Vergleich der Textabschnitte heranzuziehen. Mit dem Nachteil, dass die Software kein semantisches Verständnis besitzt, um genauso leicht ein Plagiat zu erkennen, wie ein Mensch.

Die softwarebasierte Plagiatanalyse muss sich folgenden Herausforderungen stellen:

- die Suche geeigneter Referenzdokumente in einer großen Dokumentkollektion
- die detaillierte Analyse der Dokumentabschnitte von Referenz und Quelle, um ähnliche Textabschnitte zu identifizieren
- die hinreichende Entscheidbarkeit eines Plagiatsverdachts

Das Ziel ist es, mit einer Software die Erkennensleistung des Menschen so gut es geht nachzubilden und dabei eine hohe Genauigkeit der Plagiatanalyse zu erzielen. Damit sind spezialisierte Algorithmen des *Information-Retrieval* verbunden, die mit den Dokumentmengen umgehen können und die Grundlage einer Plagiatanalysesoftware bilden. Die Forschung hat in diesem Bereich bereits Strategien [BSP07] und theoretische Ansätze vorgestellt [SKS07, SM07]. Ein Softwaresystem, dass diese Probleme im Detail lösen und gleichzeitig einen hohen Durchsatz erzielen soll, ist sehr komplex und erfordert einen hohen Planungsaufwand für die Entwickler.

Einen Lösungsansatz bietet das Plagiatanalysesystem Picapica. Die Plagiatanalyse von Picapica ist als dreistufiger Verarbeitungsprozess konzipiert, der aus heuristischem Retrieval, detaillierter Analyse und der Nachverarbeitung der Analyseergebnisse zusammengesetzt ist. In dieser Reihenfolge wird ein verdächtiges Quelldokument verarbeitet, beginnend mit dem Retrieval passender Referenzdokumente zum Quelldokument. Dabei erfolgt die Filterung der Referenzen über die Abfrage eines Index, der Dokumente anhand markanter Stichwörter referenziert und als Eingabe die Stichwörter des Quelldokuments erhält. Die Menge aller referenzierten Dokumente, deren Stichwörter mit denen des Quelldokuments übereinstimmen, werden als Kandidatendokumente weiterverwendet. Mit dieser Vorauswahl arbeiten die spezialisierten Algorithmen der detaillierten Analyse, um die Ähnlichkeit zwischen Referenz und Quelle festzustellen. Aufgrund der Überschneidung der Einzelergebnisse, erfolgt durch die Nachverarbeitung eine Zusammenfassung und Normalisierung, zur klaren Entscheidbarkeit und Kennzeichnung eines Plagiatsverdachts. Dieser Verarbeitungsprozess erfordert ein robustes und vor allem skalierbares Softwaresystem. Picapica ist ein web-basiertes Informationssystem, dass aus einer Menge von Softwarekomponenten besteht. Aufbauend auf einem Verbund vernetzter Computersysteme wird eine *Message-Oriented-Middleware (MOM)* benutzt, die die einzelnen, verteilten Softwarebestandteile jedes Computersystems über eine zentrale Instanz, die *Message-Queue*, koordiniert. Die Plagiatanalyse ist über ein jobbasiertes Verarbeitungsmodell auf mehrere *Jobserver* verteilt, die die Message-Queue als Datenbank noch zu bearbeitender Teilaufgaben nutzen. Diese Teilaufgaben konzentrieren sich in den einzelnen Jobs, die jeder Jobserver zur Verfügung stellt. Zur Interaktion mit dem Benutzer und zur Visualisierung der Analyseergebnisse, wird eine dynamische Web-Oberfläche eingesetzt. Mit der Benutzung eines Web-Browser ist ein betriebssystemübergreifender Zugriff auf den Web-Dienst Picapica möglich. Dieser stellt sich als rein ergebnisorientierte Schnittstelle zur Plagiatanalyse dar: Der Benutzer muss lediglich sein Dokument hochladen und auf das Analyseergebnis warten, um es auszuwerten. Die oben genannten Schritte der Plagiatanalyse sind vollständig automatisiert und transparent für den Benutzer. Nun besteht allerdings die Tatsache, dass nicht alle Benutzer den öffentlichen Dienst nutzen wollen oder können. Aus Gründen des Datenschutzes und zum Vertrieb in Unternehmenskreisen und an heimische Anwender, wird die Forderung nach einem portablen Picapica bekräftigt.

Ziel dieser Arbeit ist die Umsetzung dieser portablen Variante des Plagiatanalysesystems Picapica, die auf denselben Funktionalitäten des web-basierten Systems aufbaut und diese so anpasst, dass sie effizient auf einem einzigen Computersystem lauffähig sind. Mit der Identifikation der hinreichenden und notwendigen Bestandteile des Softwaresystems ist die Basis geschaffen, die notwendigen Kriterien für eine plattformübergreifende Portabilität anzulegen. Dazu wird im Vorfeld dem angestrengten Planungsaufwand der Entwickler von Grund auf nachgegangen, um die Intentionen der Entwickler bei der Verwendung bestimmter Mittel und Methoden der Softwareentwicklung kenntlich zu machen. Erst über das grundlegende Verständnis des Systemmodells, wird die Absicht hinter der Softwarearchitektur und Organisation von Picapica deutlich. Damit wird die Entwicklung des Plagiatanalysesystems Picapica reflektiert und die ausschlaggebenden Merkmale der Software charakterisiert, um sie im Kontext des portablen Picapica zu betrachten.

Für ein grundlegendes Verständnis des Softwaresystem müssen die essentiellen Konzepte

und Richtlinien der Softwareentwicklung betrachtet werden. Diese sind im Hinblick auf die Entwicklung von Picapica im Abschnitt 2.1 aufgeführt, gefolgt von einer Übersicht häufig verwendeter Architekturen in Abschnitt 2.2, die ebenfalls bei Picapica Anwendung finden. Abschnitt 2.3 führt den Blick hin zum komponentenorientierten Softwaredesign. Im weiteren Verlauf werden im Abschnitt 2.4 die Grundlagen für einen portablen Softwareentwurf erörtert. Damit münden die Erläuterungen zu den teilweise abstrakten Konzepten der Softwareentwicklung, in der Beschreibung des Verarbeitungsprozesses der Plagiatanalyse (Abschnitt 3.1). Aus diesem konzeptuellen Aufbau ist ein Softwaresystem erwachsen (Abschnitt 3.2), dass unter anderem geprägt ist von einem komponentenorientierten Softwaredesign, wie in Abschnitt 3.4 ersichtlich. Als web-basiertes Informationssystem verinnerlicht Picapica eine benutzerfreundliche Web-Oberfläche, die im Abschnitt 3.3 beschrieben wird. Zu guter Letzt führt Abschnitt 3.5 das Augenmerk auf das portable Plagiatanalysesystem.

2 Softwarearchitektur und Entwicklung

Die Entwicklung von Softwaresystemen ist ein Bereich stetiger Veränderung. Von Anbeginn der Softwareentwicklung haben sich verschiedene Vorgehensweisen in unternehmensspezifischen sowie freien Projekten herauskristallisiert. Aus diesen gewachsenen Ansätzen, etablierten sich etliche als empfohlene Richtlinien im Entwurfs- und Implementierungsprozess. Der bis heute wachsenden Zahl an Richtlinien wollen viele Publikationen gerecht werden, indem sie spezielle Teilaspekte der Softwareentwicklung behandeln. Von der Softwaremodellierung, über die Implementierung, bis hin zur Qualitätssicherung liefern sie Prinzipien für konkrete oder abstrakte Anwendungsszenarien. Andere versuchen, durch allgemeine Vorgehensweisen und Richtlinien zum Softwareengineering, die Menge an Strategien zusammenzufassen. Dieser Abschnitt wird ebenfalls Grundlagen und Paradigmen der Softwareentwicklung beleuchten. Das Hauptaugenmerk liegt dabei jedoch auf den Vorbereitungen und Überlegungen zum Plagiatanalysesystem Picapica. Hier soll der Sichtweise der Entwickler selbst Rechnung getragen werden, indem konkrete Anforderungen, mit allgemeingültigen Prinzipien verwoben und erläutert werden.

2.1 Paradigmen der Softwareentwicklung

Grundlage jeder Softwareentwicklung ist die Modellierung des Softwaresystems. Ein Systemmodell bildet dabei eine klare, von der Umwelt abgegrenzte Gesamtheit von Systemelementen ab, die untereinander in Beziehung stehen. Damit ist das Systemmodell die Abbildung eines Ausschnitts der Realität unter bestimmten Aspekten. Diese Aspekte zeigen sich als konkrete Problemstellungen, zu deren Lösung die Software entwickelt werden soll. Das Systemmodell, als abstrakte Abbildung dieser Problemstellungen, dient dazu, die charakteristischen Merkmale des Softwaresystems erkennbar zu machen. Dabei unterstützen drei wichtige Paradigmen die Modellierung eines Softwaresystems, indem sie die Komplexität des betrachteten Ausschnitts reduzieren. Diese sind

- Abstraktion,
- Softwarestrukturierung und
- Modularisierung.

Abstraktion, Softwarestrukturierung und Modularisierung sind die wichtigsten Vertreter einer ganzen Reihe von bekannten Paradigmen in der Softwareentwicklung. Sie sind essentiell in Bezug auf die Entwicklung des Plagiatanalysesystems Picapica und grundlegend für die nachfolgenden Ausführungen.

2.1.1 Abstraktion

Abstraktion bezeichnet die Verallgemeinerung eines realen Sachverhaltes zum Zweck der Abbildung in einem Modell. In Sinne der Verallgemeinerung wird dieser Sachverhalt in seine Teilaspekte zerlegt und allmählich von spezifischen Einzelheiten befreit. Dabei werden unwesentliche Bestandteile ausgeblendet, um wesentliche Merkmale hervorzuheben. Eine allgemeine Betrachtung wird möglich und im Idealfall treten die gemeinsamen Merkmale der Teilaspekte hervor. Dadurch vereinfacht sich das Verständnis des zugrunde liegenden Sachverhaltes, indem seine wesentlichen Charakteristika im Mittelpunkt stehen. Das Systemmodell bedient sich dieses abstrakten Ansatzes, für die Definition und Anordnung seiner Elemente. Dabei ist der Abstraktionsgrad der Elemente wesentlich für das Modell. Dieser äußert sich in einer unterschiedlich starken Ausprägung der Abstraktion. Einige Elemente stellen einen allgemeinen Zusammenhang der Teileaspekte dar, während andere einen konkreten Zugang zu einem Problem charakterisieren. Umso weiter also das Modell eines Sachverhaltes von seiner realen Ausprägung entfernt ist, umso höher ist der Grad der Abstraktion. Daher wird auch oft der Begriff der Abstraktionsebenen verwendet, um die Abstufungen der Abstraktion zu bezeichnen. Ein ausgewogenes Maß zwischen Abstraktion und dessen Gegenteil Konkretisierung ist wichtig, um Softwaresysteme für eine Reihe von Aufgabenfeldern zu modellieren. Infolge dessen fällt die Spezialisierung der Systeme hin zu Einzellösungen einfacher. Die softwaretechnische Ausgestaltung der Modellansätze wird verständlicher und einfacher. Abstraktion kennzeichnet sich zusammenfassend durch:

- ausblenden des Konkreten und Unwesentlichem
- aufzeigen der wesentlichen, gemeinsamen Merkmale

Abstraktion trägt zum Verständnis und der geeigneten Zusammenfassung konkreter Sachverhalte zu modelltechnischen Systemelementen bei. Die darauf aufbauende Anordnung dieser Elemente zueinander ist ein nicht zu vernachlässigender Punkt.
[Bal98, S. 559ff][Inf08]

2.1.2 Softwarestrukturierung

Die Anordnung der Systemelemente in geordneten Zusammenhängen ist Aufgabe der Softwarestrukturierung. In jedem Softwaresystem ist die Struktur des Ganzen ausschlaggebend für das Verständnis und die Benutzung seiner Einzelteile. Die Softwarestruktur bildet ein Ordnungsgefüge der wechselseitig abhängigen Systemelemente (Abbildung 2.1). Durch die unterschiedlichen Abstraktionsgrade dieser Elemente treten gerichtete Abhängigkeiten auf. Dadurch kann eine Rangordnung unter den Systemelementen definiert werden. Infolge dessen, wird durch eine definierte Rangordnung das Systemmodell hierarchisch organisiert. Diese Hierarchie stellt implizit eine restriktive, gerichtete Softwarestruktur dar. Damit ist die Anordnung der Systemelemente festgelegt durch die Softwarestruktur, abhängig von der Abstraktion der Elemente.

Abbildung 2.1: Softwarestrukturen ordnen Elemente (Punkte) zu einem Beziehungsgefüge an.

Treffend beschreibt Kopetz den Begriff der Struktur in der Softwareentwicklung unter Einbezug der Abstraktion, wie folgt:

> "Unter der Struktur eines Systems versteht man die reduzierte Darstellung eines Systems, die den Charakter des Ganzen offenbart. Losgelöst vom untergeordneten Detail beinhaltet die Struktur die wesentlichen Merkmale des Ganzen."[Kop76, S. 39]

Durch die Struktur eines Softwaresystems, wird dessen Organisation und funktionaler Zusammenhang deutlich. Folglich reflektiert sie die Architektur der Software. Sie ist zentrale Voraussetzung für klare Linien in Verarbeitung und Verhalten eines Softwaresystems. [Bal98, S. 567ff][Inf08]

In Anbetracht großer Softwareprojekte, werden die Vorteile von Softwarestrukturen erkennbar. Stetige Schwankungen in den Entwicklerteams – wie es häufig beim Picapica-Projekt der Fall ist – erfordern einen erhöhten Informationsaufwand zur Einarbeitung neuer Projektteilnehmer. Die folgenden Eigenschaften eindeutiger Softwarestrukturen erleichtern den Informationsbedarf:

- Beherrschbarkeit der Komplexität des Systems,
- erleichterte Verständlichkeit funktionaler Zusammenhänge,
- bessere Einarbeitung in fremde Softwaresysteme.

Von der abstrakten, organisatorischen und funktionalen Modellierung eines Softwaresystem folgt der nächste Schritt hin zur logischen, modularen Aufteilung.

2.1.3 Modularisierung

Das Prinzip der Modularisierung ist eine etablierte Methodik in der Softwareentwicklung. Voraussetzungen sind in erster Linie geeignete Softwaremodelle und damit verbundene, verständliche Softwarestrukturen. Nur so können die nachfolgenden Anforderungen eines Moduls vernünftig umgesetzt werden.

Module bilden funktionale Einheiten
: Das Softwaresystem wird in separate, funktionale Einheiten aufgesplittet, die zusammengehörige Systemelemente vereinen.

Module sind überwiegend kontextunabhängig
: Die funktionale Einheit ist in sich abgeschlossen, d.h. alle zur Problemlösung nötigen Funktionen und Informationen befinden sich an einer Stelle. Dadurch werden Abhängigkeiten zur Modulumgebung minimiert. Dieses als Lokalitätsprinzip bekannte Vorgehen, beinhaltet die Trennung relevanter Informationen von unwesentlichen.Nicht relevante Informationen müssen in andere Module verschoben werden. Unter Einhaltung dieses Prinzips ist eine stark differenzierte Aufteilung der funktionalen Einheiten notwendig. Diese Aufteilung bestimmt den Grad der Lokalität für das Modul. Das heißt, es muss genau bestimmt werden, wie viel Funktionalität zum Modul gehören soll, ohne unnötige Kopien in anderen Modulen zu erzeugen.

Module besitzen definierte Schnittstellen
: Um mit anderen Modulen in Interaktion treten zu können und eigene Abhängigkeiten durch Benutzung anderer Module zu lösen, werden Schnittstellen benutzt. Ein Modul definiert eine festgelegte Schnittstelle nach Außen, die von Anwendern oder Modulen benutzt werden kann. Dabei wird nur die Schnittstelle zur Verfügung gestellt, im Idealfall soll der Einblick in die internen Programmstrukturen verwehrt bleiben. So werden nur zur Benutzung relevante Informationen sichtbar. Damit verhindert die Definition einer Schnittstelle eine angepasste Programmentwicklung aufgrund des Wissens über die Modulinterna. Diese Methodik ist bekannt als Geheimnisprinzip, mit der Zielsetzung ein generisches, wiederverwendbares Modul zu konstruieren.

Aufbau und Konzeption von Modulen erfordern ein genaues Verständnis des Softwaresystems, seiner Organisation und geplanten Funktionalität. Mit einer eindeutigen und klaren Softwarestruktur, wird die Sicht auf das System wesentlich vereinfacht. Die Iden-

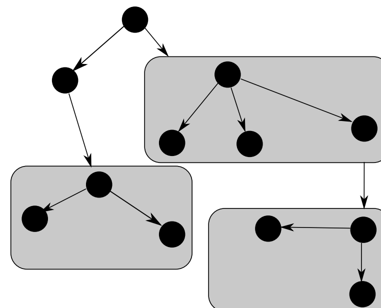

Abbildung 2.2: Gruppierung von Elementen (Punkten) in Modulen (Boxen).

tifikation zusammengehöriger Elemente der Software wird deutlich und damit auch ihre Gruppierung zu einem Modul (Abbildung 2.1.3). Durch die Softwarestruktur wird bereits eine Abstraktionsebene vorgegeben indem der allgemeine, funktionale Zusammen-

hang der Einzelelemente abgebildet wird. Durch die Gruppierung der Systemelemente bilden Module eine weitere Stufe der Abstraktion. Damit ist die Modularisierung auch immer verbunden mit der Bildung von Abstraktionsebenen, also bestimmten Abstufungen der Abstraktion. So können Module ganz konkret auf einen bestimmten Problemteil hin aufgebaut werden oder durch eine höhere Abstraktionsebene generischer ausfallen. Denn ein Modul abstrahiert vor allem durch das oben erwähnte Schnittstellenprinzip seine "innere" Funktionalität. Wird diese Schnittstelle soweit verallgemeinert, dass sie über Parameter auch in anderen Zusammenhängen benutzt werden kann, ist die Rede von einem generischen, wiederverwendbaren Modul. Dieser Aspekt der Wiederverwendbarkeit ist entscheidend, um das Prinzip der Modularisierung einzusetzen. So ist es im Nachhinein möglich, einmal entworfene Module über Parametrisierung für unterschiedliche Kontexte zu spezialisieren. Durch die angesprochene Kontextunabhängigkeit können Module unabhängig entwickelt und gewartet werden. In Verbindung mit einer wohlüberlegten und funktionalen Zusammenstellung, bleiben sie überschaubar und verständlich. [Bal98, S. 571ff, 574ff, 576ff][Kü94, S. 42-44]

2.1.4 Softwarequalität

Zusätzlich zu den drei Paradigmen der Systemmodellierung, liegt ein wichtiger Schwerpunkt auf dem Systemverhalten und der Qualität der Software. Die folgenden wichtigen Gesichtspunkte erläutern diese Aspekte der Softwareentwicklung.

Modifikation und Wartung
: Modifikation und Wartung gehören zum Prozess der Softwareentwicklung und sind wichtig für das Fortbestehen und die kontinuierliche Verbesserung eines Softwaresystems. Neue wie alte Entwickler sind zuständig für die Wartung und Weiterentwicklung eines Softwaresystems. Dadurch besteht seitens der neuen Entwickler ein hoher Lernaufwand für ein Verständnis des Systems. Genau hier greifen die erwähnten Prinzipien. Eine gute Strukturierung erleichtert die Einarbeitung neuer Entwickler und verbessert die Verständlichkeit des Systems. Modularisierung erlaubt die Zuweisung spezifischer Arbeitsfelder, ohne explizites Vorwissen anderer Systembestandteile. Infolge dessen, wirken sich Änderungen an einzelnen Modulen nicht auf die Stabilität des Systems aus, solang die öffentlichen Schnittstellen unverändert bleiben. Weiterhin gehört eine aussagekräftige Dokumentation der implementierten Funktionalität – in allen Teilen des Projektes – dazu. In gleicher Weise ist die Verwendung einer eindeutigen, auf den Kontext abgestimmten Namensgebung im Quellcode wichtig für das Verständnis. Nachfolgende Entwickler werden in jedem Fall davon profitieren und so die Einarbeitungsphase kürzer gestalten.[Bal98, S. 558-580]

Stabilität, Robustheit und Skalierbarkeit
: Stabilität, Robustheit und Skalierbarkeit sind Stichwörter, die die häufigsten Zielsetzungen in der Softwareentwicklung wiedergeben. Sie können als allgegenwärtig im Entwicklungsprozess und der nachfolgenden Softwarequalitätssicherung bezeichnet werden. Auch bei Picapica spielen sie eine ausschlaggebende Rolle. Förderlich sind maßgeblich klare Strukturierungen und angemessene Abstraktionsgrade der Systemelemente. Standardisierte Schnittstellen und klare Aufgabentren-

nung, wie sie die Modularisierung empfiehlt, tragen zu robusten Systemen bei. Auf diese Weise kann an mehreren Modulen gleichzeitig gearbeitet werden, ohne die Gesamtstabilität des Systems zu gefährden.

Diese Prinzipien und Vorgehensweisen sind ein Auszug aus einer ganzen Reihe von empfohlenen Richtlinien, für das Arbeitsfeld der Softwareentwicklung. Sie werden häufig nur am Rande betrachtet oder ganz vernachlässigt, was sich im Nachhinein als fatal auswirken kann. Nachträgliche Änderungen am Projekt sind aufwendig und erfordern meist sowohl die Reorganisation der Projektteams als auch der Entwicklungsprozesse. Aus diesem Grund ist ein gut organisiertes Systemmodell im Vorfeld der Softwareimplementierung entscheidend für den Werdegang eines Softwaresystems. Viele freie und kommerzielle Entwicklungsprogramme wollen diesen Entwurfsprozess unterstützen, doch das grundlegende Systemmodell wird mit Modellierungssprachen, wie UML (Unified Modeling Language), aufgebaut und durch diese Sprachen finden die genannten Richtlinien ihre Anwendung [Wik08c]. Die betrachteten Prinzipien bilden für die Entwickler von Picapica die Quintessenz einer Menge von Empfehlungen. Durch ihre konsequente Umsetzung lassen sie sich als Faustregeln für das Picapica-Projekt bezeichnen. Eine Überlegung, ebenso wie die Anwendung dieser Richtlinien, ist in jedem Fall empfehlenswert und erleichtert die Arbeit, auch in anderen Softwareprojekten um einiges.

2.2 Softwarearchitekturen im Überblick

Im vorherigen Abschnitt wurden grundlegende Prinzipien der Softwareentwicklung erläutert, auf die im Folgenden aufgebaut wird. Softwaremodelle organisieren Systemelemente, um die wesentlichen Merkmale einer Software abzubilden. In dieser Hinsicht sind abstrakte Softwaremodelle, losgelöst von konkreten Szenarien, charakteristisch für eine ganze Reihe von Softwaresystemen. Sie zeichnen sich durch interne und externe Interaktionsmöglichkeiten und Abhängigkeiten aus. In diesem Zusammenhang charakterisieren die Abhängigkeiten zum Aufbau und der Verwendung von Systemelementen und Ressourcen, den Kopplungsbegriff. Dieser Begriff zeigt sich in den Softwaremodellen in unterschiedlicher Ausprägung. Die gebräuchlichsten und am häufigsten verwendeten Systemmodelle sind Gegenstand dieses Abschnitts:

- monolithische Software
- Client-Server-Software
- dienstorientierte Software
  - Web-Dienste

2.2.1 Monolithische Software

Monolithische Softwaresysteme vereinen ihre funktionalen Elemente in einem homogenen Gebilde. Aufgrund der starken Kopplung der Elemente, treten bei Modifikationen häufig Seiteneffekte auf. Somit bewirkt eine Änderung, eine Reihe von Folgeänderungen.

Diese Systeme sind überwiegend stark gebunden an Hardwareressourcen, interne Datenformate und proprietäre Schnittstellen. Die Wiederverwendung in anderen Anwendungsbereichen, zieht somit eine Änderung der Anwendungslogik nach sich. Folglich sind monolithische Softwaresysteme, spezialisiert auf einen bestimmten Anwendungskontext und wenig flexibel in ihrer Wiederverwendung. Ein Beispiel für ein monolithisches System ist der Linux-Kernel. Als Modul betrachtet, kapselt er die Grundfunktionalität, auf die eine Menge von Zusatzmodulen aufbauen. Diese Grundfunktionen sind stark an die Maschine und interne Hardwareressourcen gekoppelt. Um diese Abhängigkeiten zu abstrahieren, werden Schnittstellen bereitgestellt, die die Zusatzmodule zur Interaktion nutzen.

2.2.2 Client-Server-Software

Im Gegensatz zu diesen eng gekoppelten Softwaresystemen, existieren Vertreter, die auf das Prinzip verteilter Ressourcen aufbauen. Client-Server-Systeme setzen auf einen kooperativen Verbund von Systemelementen, deren Aufgaben über vernetzte Rechner verteilt sind. Hinsichtlich dieser Architektur existieren Server, die ihre Funktionalität als Dienstleistung über ein Netzwerk anbieten. Clients nutzen diese Dienste gegebenenfalls in ihrer eigenen Anwendungslogik. Zur Interaktion zwischen Client und Server werden häufig proprietäre Protokolle vereinbart, die auf beiden Seiten bekannt sein müssen. Weiterhin besteht zur Nutzung eines "Serverdienstes" häufig die Forderung nach einer bestehenden und beständigen Verbindung zwischen Client und Server. Aus diesen Gründen erfordert die Entwicklung dieser Systeme, erhebliche Aufmerksamkeit der Entwickler, sowohl beim Client als auch beim Server. Durch das Client-Server-Modell stellen sowohl Server, als auch Client Teilfunktionalitäten bereit, die im Verbund zur Erfüllung einer Aufgabe beitragen. Es können somit beliebige Rechnersysteme eingesetzt werden, auf denen die Teilfunktionalität bereitsteht. Dahingehend ist die Rollenverteilung, ob Client oder Server, in solchen Architekturen nicht festgelegt. Eine ungleichmäßige Verteilung der Funktionalität, sowie der gegenseitige Austausch von Client und Server, ist möglich. Das beste Beispiel, bei dem eine Unterscheidung von Client und Server keinen Sinn mehr macht, sind Peer-To-Peer-Systeme. Eine Client-Server-Architektur soll für eine optimale Ressourcenausnutzung der beteiligten Rechner sorgen, unter Zuhilfenahme eines Verteilungsmodells, dass Teilprobleme auf die zur Verfügung stehenden Rechner verteilt. [Bal00, S. 703]

2.2.3 Dienstorientierte Software

Das angewandte Konzept der Dienstleistung, das ein Server im Client-Server-Modell realisiert, bringt eine weitere Architektur zum Vorschein. Dienstorientierte Architekturen – Englisch: *service oriented architecture (SOA)* – bedienen sich der Philosophie *"software as a service"*. Die Forderungen, denen Dienste gerecht werden müssen, setzen sich zusammen aus:

- standardisierten Schnittstellen und Datenaustauschformaten
- öffentlichen Kommunikationswegen zur Interaktion

- ständige Verfügbarkeit über Netzwerkgrenzen hinweg
- zunehmende Unabhängigkeit von der zugrunde liegenden Betriebssystem- und Hardwareplattform

Daraus ergeben sich flexible Dienste, die in unterschiedlichen Aufgabenbereichen Anwendung finden können. In dieser Hinsicht kann jede Software mit entsprechender Umsetzung dieser Forderungen, als Dienst bereitgestellt werden. Ganz konkret ermöglicht das Dienste-Konzept die Integration alter, eventuell spezialisierter Software in neue Softwareumgebungen und stellt damit eine Methode der Wiederverwendung bereit.
Wie beim Client-Server-Modell nutzen Clients die Dienste, um die eigene Anwendungslogik zu vervollständigen. Die Grenze zwischen Client und Server ist beim dienstorientierten Ansatz sehr unscharf, denn Dienste können selbst andere Dienste benutzen, um komplexe Vorgänge durch einen Verbund zu realisieren. Von einem Dienst soll nicht mehr bekannt sein als durch seine Schnittstelle erlaubt wird. Aus diesem Grund lassen sich Dienste mit gleicher Schnittstelle gegenseitig austauschen. Daraus resultiert ein lose gekoppeltes Gefüge zwischen Client und Dienst. Die Substituierbarkeit der Dienste verringert den Ausfalls bereitgestellter Dienstfunktionalität, indem ein Dienst einer Modifikation oder Wartung unterzogen wird, ersetzt ein anderer Dienst mit gleicher Schnittstelle diesen Ausfall. Ein Client bleibt von dieser transparenten Ersetzung unberührt und damit trägt dieser Aspekt zur Ausfallsicherheit der Anwendungslogik eines Clients bei. Dienstorientierte Softwaremodelle verinnerlichen das Prinzip verteilter Funktionalität, ähnlich wie das Client-Server-Modell. [Pap08, S. 8-10]

2.2.4 Web-Dienste

Eine besondere Teilmenge der dienstorientierten Architektur ist der Web-Dienst. Web-Dienste setzen zur Bereitstellung ihrer Dienstleistung offene Internetprotokolle, wie HTTP ein. Der Datenaustausch wird mit Hilfe von standardisierten Datenformaten vorgenommen, wobei sich XML durchgesetzt hat. Offene Kommunikationsprotokolle und Datenformate bieten aufgrund ihrer Allgemeinheit und Bekanntheit, ein breites Spektrum an Anwendungen und Anpassungen. Daraus ergibt sich bei der Verwendung ein hoher Grad an Flexibilität. Die Kommunikationsprotokolle und Datenformate sind offen, standardisiert und für jede Client-Implementierung verfügbar. Zusätzlich zu den bereits genannten Interaktionsmöglichkeiten der Dienste, können Web-Dienste selbst eine Schnittstelle für menschliche Benutzer integrieren. Diese gestaltet sich häufig als web-basierte Benutzeroberfläche zur direkten Verwendung des Dienstes. Die Benutzeroberfläche bündelt transparent für den Benutzer, die Komplexität des Gesamtsystems, ermöglicht interaktive Dateneingaben und präsentiert die Ausgaben aufbereitet in einem Web-Browser. Damit existieren zwei grundlegende Varianten der Web-Dienste. Exemplarisch für beide Varianten sind bekannte Internetsuchmaschinen wie Google, MSN und YAHOO. Sie bieten sowohl eine reine anwendungsorientierte Schnittstelle, als auch eine aufbereitete, web-basierte Oberfläche. Die darunter liegende Server- und Systemstruktur bleibt für den Anwender transparent. [Pap08, S. 4-17]

2.3 Komponentenorientiertes Softwaredesign

Die Erfolgsgeschichte des objektorientierten Softwareentwurfs begann in den frühen 1960er Jahren mit der Einführung der Programmiersprache Simula. Bis heute werden diese Konzepte in mehreren Programmiersprachen unterstützt und weiterentwickelt, in Modellierungssprachen, wie UML, eingesetzt und als Grundprinzip guten Softwaredesigns angesehen. Somit ist die Objektorientierung Teil des Entwicklungsprozesses einer Software, vom Softwaremodell bis zur Implementierung. Die Eingangs erwähnten Paradigmen der Softwareentwicklung tragen mit der Objektorientierung, zur Ausgestaltung konkreter Verfahren der Implementierung und Modellierung bei. Als Grundlage dient die Abstraktion bestimmter Sachverhalte, durch die Elemente der objektorientierten Softwareentwicklung. Die Klasse, als objektorientiertes Elemente, definiert die Abbildung eines Sachverhaltes mit seinen Eigenschaften und Verhaltensweisen. Ein Objekt ist eine konkrete Ausprägung einer Klasse, eine Instanz. Es stellt Eigenschaften und Verhalten durch Attribute und Methoden bereit. Die Objektorientierung basiert auf den Beziehungen zwischen Objekten, ihren Interaktionen und Strukturen.
Als Vorbereitung für das komponentenorientierte Programmdesign, dient in erster Linie die Objektorientierung. Aus diesem Grund erläutert dieser Abschnitt relevante Konzepte der Objektorientierung, um zum komponentenorientierten Softwaredesign überzuleiten. Die primären Konzepte sind in dieser Hinsicht:

- Vererbung,
- Polymorphismus und
- Schnittstellen.

Vererbung

Das Konzept der Vererbung steht für die Idee, Eigenschaften, die mehreren Klassen gemein sind zusammenzufassen und damit zu "generalisieren". Die gemeinsamen Merkmale der Klassen werden in einer übergeordneten Klasse (Basisklasse) gesammelt und aus den anderen Klassen entfernt. Diese wiederum, werden als Unterklassen der neuen Basisklasse ausgezeichnet. Unterklassen haben typischerweise Zugriff auf alle Merkmale ihrer Basisklasse, sie erben diese Merkmale. So lassen sich auch hierarchische Klassenstrukturen aufbauen. Es ist den Unterklassen darüber hinaus möglich, die geerbten Merkmale ihrem Kontext entsprechend zu ändern und zu erweitern. In diesem Fall wird von einer Spezialisierung der Unterklassen gesprochen. Ein Beispiel ist in Abbildung 2.3 ersichtlich. [Bal00, S. 200]

Polymorphismus

Aufgrund der Vererbung ist es wichtig, den Begriff des Typs einer Klasseninstanz anzusprechen. Unterklassen erben neben den Merkmalen auch den Typ der Basisklasse. Dadurch kann eine Unterklasse zusätzlich zu ihrem eigenen Typ, als Typ der Basisklasse benutzt werden, jedoch nur mit der Schnittstelle, die diese zur Verfügung stellt. Erst zur Laufzeit des Objektes würde entschieden werden, welchem konkreten Klassentyp

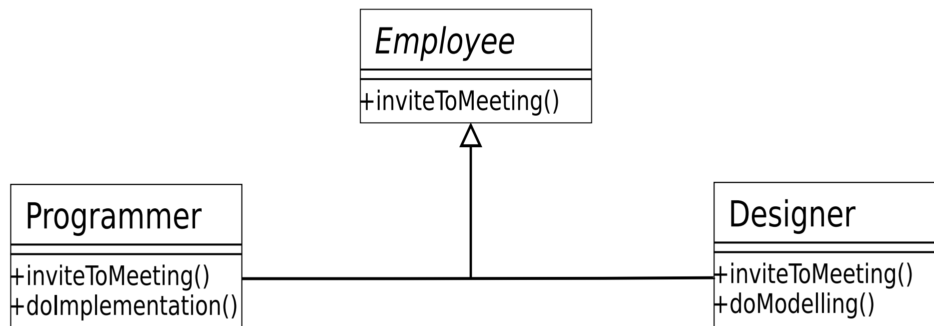

Abbildung 2.3: Beispiel für eine Vererbungshierarchie, modelliert in UML. Die Klassen *Programmer* und *Designer*, erben von *Employee* die Eigenschaften und Attribute. Diese noch inhaltslosen Merkmale, werden von jeder Unterklasse spezifisch belegt. Zusätzlich erweitern beide die Eigenschaften um spezifische Methoden.

der Aufruf zugeordnet wird. Dieses Prinzip ist als Polymorphismus bekannt und drückt die Vielgestaltigkeit der Klassen in einer durch Vererbung entstandenen Hierarchie aus. Am Beispiel von Abbildung 2.4 wird das Prinzip verdeutlicht. [Bal00, S. 828-830]

Schnittstellen

Ein weiterer Schritt, hin zum komponentenorientierten Softwaredesign, ist die Modularisierung. Module sind im Sinne der Objektorientierung vergleichbar mit Klassen oder Gruppierungen von Klassen. Diese Funktionseinheiten, die über definierte Schnittstellen verfügen, lassen sich durch assoziative Module austauschen. Dabei erlauben Schnittstellen die formale Deklaration von Methoden, ohne Kenntnisse über die Implementierung nach Außen zu reichen. Schnittstellen sind ein wesentliches Prinzip für die einheitliche Interaktion zwischen Objekten. Sie lassen sich als Erweiterung der Vererbung verstehen. Eine Klasse kann mehrere Schnittstellen implementieren und dadurch unterschiedliche Eigenschaften spezialisieren. Damit können Klassen die Eigenschaften vieler Schnittstellen "erben", was in dieser Hinsicht als Mehrfachvererbung oder Mehrfachimplementierung bezeichnet wird. Diese Möglichkeit wird von der "einfachen" Vererbung nicht unterstützt. In diesem Sinne gehören Schnittstellen nicht zur Vererbungshierarchie einer Klassenstruktur, da sie eine separate, abstraktere Einheit zu den Klassen darstellen. Sie definieren vielmehr eine Art Nutzungsvertrag für andere Benutzer, die sich häufig als andere Klassen charakterisieren. Das Prinzip der Schnittstellen ist in Abbildung 2.5 noch einmal veranschaulicht. [Bal00, S. 817-818]

Diese Konzepte der Objektorientierung sind grundlegend für moderne Modellierungs- und Implementierungsansätze. Die Softwareentwicklung setzt seit der Einführung der

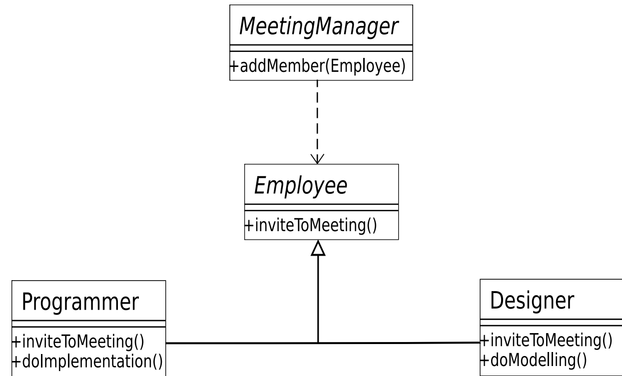

Abbildung 2.4: Die Klasse *MeetingManager* besitzt eine Methode, die den Typ der Basisklasse *Employee* benötigt. Welche Instanzen im Endeffekt am Meeting teilnehmen, wird erst zur Laufzeit festgestellt. Es können sowohl *Programmer*, als auch *Designer* sein, die über den Typ der Basisklasse dem Meeting hinzugefügt werden.

Objektorientierung zunehmend darauf auf. In unterschiedlichen Softwareprojekten werden anwendungsspezifische Sachverhalte mit objektorientierten Methoden funktional beschrieben, implementiert und miteinander kombiniert. Dabei liegt der Schwerpunkt der Entwicklungen vorrangig auf dem projektinternen Gebrauch der Funktionalität. Das bedeutet im Umkehrschluss, dass die Verwendung externer, projektfremder Funktionalität entweder einen zusätzlichen Implementierungsaufwand zur spezifischen Integration in die eigene Software bedeutet, oder eine Neu-Implementierung vorgezogen wird. Ein Technologietransfer im Sinne der Wiederverwendung etablierter Funktionalität ist hier nicht zu sehen. Das ist in letzter Konsequenz den objektorientierten Konzepten geschuldet. In ihrer Umsetzung ist die Objektorientierung fähig, die im Kapitel 2.2 angesprochenen Softwaresysteme zu realisieren. Allerdings sind diese Konzepte zu starr und unscharf, um die Entwickler bei den gewachsenen Ansprüchen agiler Softwaresysteme zu unterstützen; die Literatur gibt dazu mehr Anhaltspunkte. Die Minimierung der Entwicklungs- und Wartungskosten, sowie die schnelle Anpassung der Softwareprodukte, ist relevant für die Wirtschaftlichkeit einer Software. In dieser Hinsicht sollen im Vorfeld festgelegte Designentscheidungen zum Softwaresystem nicht revidiert, sondern vielmehr durch zusätzliche Softwarebausteine erweitert werden.

Der Trend geht hin zu abstrakteren Elementen, als Klassen oder Module, indem diese Elemente

- kontextfrei und unabhängig von ihrer Umgebung sind,
- Funktionalitäten über Schnittstellen domänenspezifisch spezialisieren,
- in jeder Software einsetzbar und wiederverwendbar sind, und
- durch ihre Funktionalität, einen Sachverhalt vollständig abdecken.

Komplexe Softwaresysteme bestehen aus einer Vielzahl einzelner Elemente, die zu eigenständigen Softwarekomponenten zusammenzufassen sind. Software muss flexibel sein

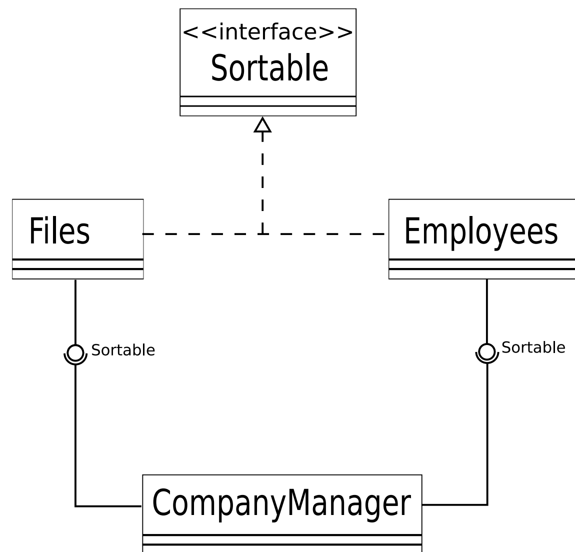

Abbildung 2.5: In diesem UML-Beispiel, wird das Prinzip der Schnittstelle verdeutlicht. Die Klassen *Employees* und *Files* implementieren die Schnittstelle *Sortable* und stellen sie anderen Klassen zur Verfügung. Hier ist es die Klasse *CompanyManager*, die die Methoden der "Sortable" Objekte später benutzt, um Ordnung zu schaffen.

und über dynamische Techniken verfügen, um Funktionalität und Schnittstellen zu erweitern. Dabei spielen die zunehmende Vernetzung und Hardwareentwicklung der Rechnersysteme eine gewichtige Rolle. Softwaresysteme setzen sich in dieser Hinsicht aus verteilten und interagierenden Softwarekomponenten zusammen. Mit der Modularisierung ist dafür ein Konzept der Softwareentwicklung gegeben, das ausschlaggebend für den Komponentenbegriff ist. Modularisierung vereint grundlegende Prinzipien für die Organisation , die Interaktion und den strukturierten Aufbau separater Funktionseinheiten. Damit baut der Komponentenbegriff auf objektorientierte Klassen und Module auf, die nachfolgend entsprechend differenziert werden müssen. Die vielen Ungereimtheiten in der Literatur lassen keine klare Definition von Komponenten erkennen, als vielmehr Erklärungsansätze. Aus diesem Grund, wird der Komponentenbegriff in diesem Abschnitt vom Klassenbegriff über den Modulbegriff hergeleitet. Im Allgemeinen können Komponenten als weitere Abstraktionsebene über Klassen und Modulen verstanden werden, inwiefern dieser Grad aufgeweicht wird, hängt von der jeweiligen Sichtweise der Entwickler ab. [Gri98, S. 1-15][Kü94, S. 26-36][Wik08a, Wik08b]

Klassen

Eine Klasse verkörpert im konventionellen, objektorientierten Ansatz die Verdinglichung eines Sachverhaltes der realen Welt. Dazu können sowohl Gegenstände als auch Handlungen und Beziehungen zählen. Zur Behandlung dieser Sachverhalte stellt die Klasse entweder eigenständig die volle Funktionalität bereit oder durch eine Kombination mit anderen Klassen.

Module

Funktionale Einheiten, die aus einer oder mehreren Klassen zusammengesetzt sind, werde im Sinne der Modularisierung als Module bezeichnet. Diese bieten definierte Schnittstellen zu einzelnen Operationen und Eigenschaften ihrer Funktionalität. Weiterführend ist der Begriff der Komponente eine Zusammenstellung eines oder mehrerer Module. Dieser Umstand, wird in der Literatur allerdings oft gleichbedeutend verwendet wird.

Komponenten

Hinter Komponenten stehen besondere softwaretechnische Anforderungen. Sie sollen eigenständige, fachlich isolierte Softwareelemente definieren, die für einen bestimmten Problembereich konfiguriert werden können. Ausschlaggebend ist hier die vollständige Abdeckung dieses Bereichs, durch die innere Funktionalität der Komponente. Hervorzuheben ist das Prinzip der Kapselung, dass nur mit der Verwendung des Schnittstellenbegriffs vollständig umgesetzt werden kann. Im Idealfall bildet eine Komponente ein abgeschlossenes, softwaretechnisches Gebilde mit konfigurierbarer Funktionalität. Diese kann ad hoc – als aktive Komponente – verwendet werden oder im Zusammenhang mit anderen Komponenten – als passive Komponente – die geforderte Funktionalität bewerkstelligen. So wird der Weg deutlich, den die Softwareentwicklung zukünftig beschreiten wird:

> "Wähle die richtige Komponente und verwende dann, was sie Dir bietet. [...] Programme wandeln sich daher zunehmend von einer monolithischen Aneinanderreihung einzelner Prozeduraufrufe *(action-oriented)* hin zu Collagen interagierender Softwarekomponenten *(component-based)*." [Gri98, S. 19, 21]

Für konfigurierbare Komponenten ist eine wohlüberlegtes Maß an Funktionalität und Abstraktion notwendig:

> "Eine Komponente ist ein Stück Software, das klein genug ist, um es in einem Stück erzeugen und pflegen zu können, groß genug ist, um eine sinnvoll einsetzbare Funktionalität zu bieten und eine individuelle Unterstützung zu rechtfertigen sowie mit standardisierten Schnittstellen ausgestattet ist, um mit anderen Komponenten zusammenzuarbeiten." [Gri98, S. 31]

Daraus ergeben sich wichtige Kriterien für das Komponentendesign, die es zu beachten gilt. Diese charakterisieren sich im Folgenden als die Stichwörter

- Wiederverwendung,
- Granularität,
- Schnittstellen und
- Kopplung.

Wiederverwendung

Maßgeblich für den Einsatz von Komponenten ist ihre Wiederverwendbarkeit. Um den zusätzlichen Entwicklungsaufwand, des komponentenorientierten Softwaredesigns zu wagen, sind im Vorfeld softwaretechnische Überlegungen notwendig. Dem Entwurf von Komponenten und damit auch Applikationen, geht die Entscheidung bestimmter Vorgehensweisen voraus. Soll die Komponente auch in einem anderen Kontext verwendet werden oder verwendet sie andere, bereits vorhandene Komponenten und erzeugt dadurch externe Abhängigkeiten. Damit liegt die Wahl bei der Entwicklung für eine Wiederverwendung oder der Entwicklung mit Wiederverwendung. Die Entwicklung für eine domänenunabhängige Wiederverwendung wird als *Domain-Engineering* bezeichnet. Die Entwicklung mit Wiederverwendung anderer Komponenten des betrachteten Kontextes, nennt sich *Application-Engineering*. Beide Überlegungen erfordern bestimmte Prämissen für den aktuellen Aufgabenbereich. Einerseits die Anpassung und Integration vorhandener Komponenten, andererseits die Identifikation funktionaler Charakteristika des geplanten Aufgabenbereichs und ihrer Bedeutung für eine domänenunabhängige Verwendung. Damit ist die Entwicklung einer wiederverwendbaren Komponente, von Entscheidungen über deren Wiederverwendbarkeit geprägt. [Gri98, S. 15ff, 48-50]

Granularität

Welche Funktionalität in einer Komponente bereitgestellt wird und wie sich diese im Verbund mit anderen verhält, kennzeichnet die Granularität der Komponente. Dabei sind Entscheidungen zur strukturellen Abgrenzung und Modellierung dieser Funktionalität, als eigenständiges Element, richtungsweisend. Die notwendigen Analysen des Aufgabenbereichs fallen in das Aufgabenfeld des erwähnten *Domain-Engineering*. Maßgeblich für die Zusammenstellung ist die Fähigkeit von Komponenten, miteinander kombinierbar zu sein. So entstehen bei zu kleiner Funktionalität viele Abhängigkeiten bzw. Kopplungen zu anderen Komponenten, die miteinander kombiniert werden müssen. Indes führt eine zu große Funktionalität einer Komponente, zu verminderten Kombinationsmöglichkeiten mit anderen, die Wiederverwendung und Weiterverwendung in anderen Kontexten wird also erschwert. Die Begriffe aktive und passive Komponente sind bei diesen Überlegungen erneut von Bedeutung. Die Analyse und Auswertung dieser Ansätze schlägt sich später in der Bereitstellung der Komponenten nieder. Hierbei ist die Verwendung durch andere Benutzer bzw. Komponenten und die Integration in eigene Applikationsstrukturen gemeint. In jedem Fall muss eine eindeutige, definierte Zugriffsmöglichkeit für die realisierte Komponente sichergestellt werden. [Gri98, S. 50-53]

Schnittstellen

Die Schnittstelle einer Komponente, ist von zentraler Bedeutung im komponentenorientierten Programmdesign. Der Schnittstellenbegriff sorgt für den äußeren Rahmen der Komponente, ihrer definierten Abgrenzung zur Umwelt. Damit ist die Schnittstelle einer Komponente bezeichnend für ihre Interoperabilität und Interaktionsfähigkeit. Sie listet die Menge an zugänglichen Eigenschaften auf, die eine Komponente nach Außen bereitstellt. Der somit bezeichnete Vertrag der Softwarekomponente, den Entwickler eingehen müssen, realisiert die Rahmenbedingungen für eine einheitliche Benutzung. Die bereits genannte Möglichkeit der Mehrfachimplementierung, trifft auf Komponenten in gleicher Weise zu wie auf Klassen. Dadurch ergeben sich für jede bereitgestellte Schnittstelle unterschiedliche Sichten auf die Komponente. Weiterhin lassen sich Komponenten mit gleicher Schnittstelle gegeneinander austauschen, wie es schon bei den dienstorientierten Architekturen angemerkt wurde. Damit ist die Schnittstelle als zentraler Zugriffspunkt einer Komponente, durch ihren Abstraktionscharakter maßgeblich für die Interaktionsfähigkeit der Komponente und den Zusammenbau eines komponentenbasierten Gesamtsystems. Bei allen Möglichkeiten, die die Komponentenschnittstellen bieten soll natürlich auf die Eigenständigkeit einer Komponente geachtet werden. Dabei gilt es, die innere Kopplung der Komponente zu maximieren und die äußere Kopplung zu minimieren. Die innere Kopplung einer Komponente, bezieht sich auf die interne Funktionalität, ist damit Gegenstand der erwähnten Betrachtungen zur Granularität. [Gri98, S. 53-68]

Kopplung

Ausschlaggebend für die Wiederverwendung einer Komponente, sind ihre Abhängigkeiten zur Umwelt, ihre äußere Kopplung. Dazu zählen auch programmiersprachliche Einschränkungen, denn nicht jede objektorientierte Programmiersprache setzt das Schnittstellenkonzept vollständig um. Eine ideale Komponente ermöglicht durch ihre generische Beschreibung, eine eindeutige Trennung zwischen ihrer Implementierung und Schnittstelle. Dadurch soll den unterschiedlichen Einschränkungen objektorientierter Definitionen entgangen werden. Hier ist die Benutzung einer Beschreibungssprache für Schnittstellen vorgesehen, die losgelöst von jeglicher Implementierungssprache, eine abstrakte Schnittstellendefinition zulässt. Diese wird als *Interface Definition Language (IDL)* bezeichnet. Die Sprache lässt die Generierung der Schnittstellendefinition in unterschiedlichen Zielsprachen zu. Somit bleibt die Verwendung der Schnittstellen konsistent, obwohl sich die Implementierungssprachen unterscheiden können. Die sprachspezifischen Kopplungen der Komponenten sollen so auf ein Minimum gesenkt werden, was den Einsatz der Komponenten in heterogenen Systemen fördert. Diese Steigerung der Interoperabilität ermöglicht die Wiederverwendung einer Vielzahl von Komponentenbausteinen unterschiedlicher Herkunft. Die funktionale, äußere Kopplung einer Komponente ist ein weiterer Aspekt. Wichtigstes Kriterium für eine Komponente ist die minimale Abhängigkeit zu anderen Komponenten, um weitgehend, eigenständig zu funktionieren. Die Analyse und Beseitigung von Kopplungsarten, muss bereits bei der Modellierung und Funktionszuteilung einer Komponente geschehen und wird als Entkopplung bezeichnet. Dabei lassen sich die Arten *unidirektional*, *bidirektional* und *zyklisch* unterscheiden (Abbildung

2.6).

- Die unidirektionale Kopplung besagt, dass eine oder mehrere Klassen aus Komponente A abhängig von Klassen aus Komponente B sind.
- Bidirektionale Kopplung liegt vor, wenn funktionale Klassen einer Komponente abhängig von Klassen einer anderen Komponente sind und umgekehrt, nur zusammen erfüllen sie Funktionalität.
- Zyklische Kopplung tritt bei gegenseitigen Abhängigkeiten von mehr als drei beteiligten Komponenten auf.

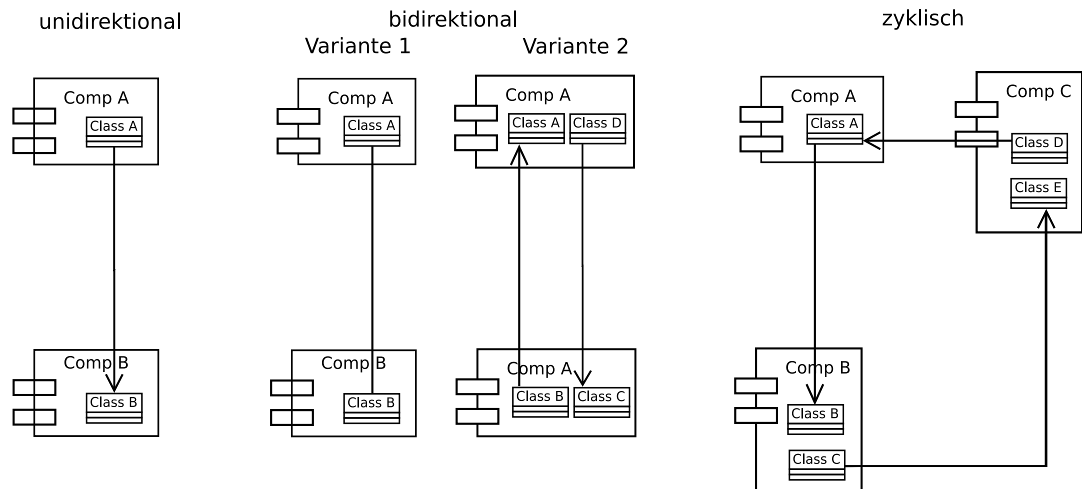

Abbildung 2.6: Kopplungsarten von Komponenten

Entkopplungstechniken sind in der Literatur beschrieben und werden zum Teil auch durch spezielle Programme ermöglicht.[GW99]

Kopplung umfasst im Komponentenparadigma nicht nur die auftretenden Abhängigkeiten zwischen Softwarekomponenten. Darüber hinaus, zählen aufgrund der Forderung nach einer systemübergreifenden Verwendung, die Abhängigkeiten zu programmiersprachlichen Modellen, Betriebssystemen und Hardwareplattformen ebenfalls zur Kopplung. Durch die minimale, äußere Kopplung soll im Idealfall eine problemlose Überführung der Komponenten zwischen unterschiedlichen Hard- und Softwareplattformen möglich sein. Der Einsatz von Komponenten soll sich dadurch, in heterogenen Systemen flexibel und dynamisch gestalten.

Der komponentenorientierte Softwareentwurf setzt auf die Komposition fertig einsetzbarer Softwarekomponenten mit abgeschlossener Funktionalität. Diese können aus einem Pool von Komponenten gewählt und durch die klar spezifizierten Schnittstellen beliebige kombiniert werden. Wiederverwendung selbst entworfener Komponenten bedeutet zugleich die Beachtung ihrer Einsatzzwecke, die Abstraktion auf generische Charakteristika

und ihrer Spezialisierung für andere Arbeitsbereiche. Damit erfordert komponentenorientiertes Softwaredesign erheblichen Mehraufwand als herkömmliche, meist objektorientierte Modellierungsansätze. Für den Softwareentwurf und die Implementierung neuer Komponenten sind Analysen zur Granularität, den Schnittstellenspezifikationen und der Wiederverwendbarkeit nötig. Diese Arbeitsschritte nehmen viel Zeit in Anspruch und sind für zeitlich begrenzte Softwareprojekte nicht ratsam. Im Allgemeinen ist beim komponentenorientierten Softwareentwurf, zwischen Komponentenentwicklern und Anwendungsentwicklern zu unterscheiden. Komponentenentwickler sind für die feinkörnige Funktionalität und ihrer Zusammenstellung innerhalb einer Softwarekomponente – also dem *programming-in-the-small* – zuständig. In anderer Weise beschäftigen sich Anwendungsentwickler mit dem konzeptuellen Aufbau der komponentenbasierten Software. Der Entwurf und die Umsetzung solcher Kompositionsanwendungen soll durch die Verwendung erprobter, fertiger Softwarekomponenten erleichtert werden. Ähnlich einem Baukastensystem bilden diese Softwarekomponenten durch eine moderate Anpassung an den aktuellen Kontext die fertige Software. Daraus resultieren weitaus schnellere Entwicklungszyklen der Programme und Programmfamilien, sowie reduzierte Entwicklungskosten. Mit der Verwendung ausgereifter Softwarekomponenten entfällt der Neuentwurf der Funktionalität. Aufgrund der spezifizierten Schnittstellen erlauben Komponenten nur eine *Black-Box-Sicht*. Damit ist die Verdeckung der zugrunde liegenden Implementierungsdetails gemeint. Im Umkehrschluss bedeutet das für die Entwickler, dass sie kein inhärentes Verständnis über die Implementierung der Funktionalität mitbringen müssen, wohl aber über deren Verwendbarkeit. Durch diese verringerte Anwendungskomplexität richtet sich der Fokus auf die wesentlichen, konzeptuellen Bestandteile der Anwendung, dem *programming-in-the-large*.

> ”Structuring a large collection of modules to form a 'system' is an essentially different intellectual activitiy from that constructing the individual modules. That is, we must distinguish Programming-in-the-large from Programming-in-the-small”
> [DK75]

Das Komponentenparadigma mit seinen Designansätzen und Implementierungsmerkmalen ist ein zunehmend bedeutsamer Bereich der Softwareentwicklung. Das grundlegende Verständnis dieser Methodik ist wichtig für den weiterführenden Blick auf das komponentenorientierte Plagiatanalysesystem Picapica.

Um das Begriffsbild der Komponente und das, einer komponentenorientierten Architektur noch einmal zu festigen, folgen einige aus der Literatur [Gri98, S. 68] bekannte Merkmale.

Trennung von Schnittstelle und Implementierung
: Damit ist die durchgängig losgelöste Spezifikation einer Komponente von ihrer Implementierung gemeint.

Plug & Play Fähigkeiten
: Diese Eigenschaft soll den sofortigen Einsatz vollständig entkoppelter Komponenten ohne Vorbedingungen ermöglichen.

Orts- und Plattformtransparenz
: Dadurch wird garantiert, dass Komponenten unabhängig von Systemumgebungen und Lokalität über vernetzte Strukturen hinweg verwendbar sind, ähnlich den erwähnten Web-Diensten.

Integrations- und Kompositionsfähigkeit
: Ein Merkmal, dass die Verwendung einer Komponente innerhalb anderer Komponenten und im Verbund, innerhalb von Kompositionsanwendungen, zusichert.

Der Weg der Softwareentwicklung ändert sich von der reinen Objektorientierung, zum komponentenorientierten Softwaredesign und den sogenannten Kompositionsanwendungen, bestehend aus vielen vorgefertigten Softwarekomponenten. In Abbildung 2.7 wird das noch einmal verdeutlicht.

2.4 Entwurf einer portablen Software

Portabilität bezeichnet im Umfeld der elektronischen Datenverarbeitung die Eigenschaft einer Software, auf unterschiedlichen Plattformen einsetzbar zu sein. Als Plattform wird der Verbund von Betriebssystemen und Hardwarekomponenten in Datenverarbeitungsanlagen oder Computern bezeichnet. Aus der Tatsache heraus, dass nicht jede bestehende Software portabel gestaltet ist, schließt Portabilität die Aufwendungen mit ein, die geleistet werden müssen, um diese Software entsprechend anzupassen. Dieser nachträgliche Entwicklungsaufwand wird als Portierung bezeichnet. Für eine erfolgreiche Portierung sind Leistungen für zwei grundlegende Prinzipien aufzubringen.

- Transport - Der Transport einer Software von einer Plattform zu einer Zielplattform über physikalisch kompatible Übertragungsmedien.
- Adaptierung - Die Adaptierung der Originalimplementierung, durch Modifikation und Angleichung an die Zielplattform.

Damit muss sowohl der Übertragungsweg einer Software als auch ihre Funktionalität, zwischen der Zielplattform und der ursprünglichen Plattform portabel sein oder portabel gestaltet werden. Im Vorfeld der Entscheidung, eine Software auf eine andere Plattform zu portieren, sollte der zu erbringende Portierungsaufwand mit einer Neuimplementierung ins Verhältnis gesetzt werden, um den Nutzen einer Portierung abzuschätzen.

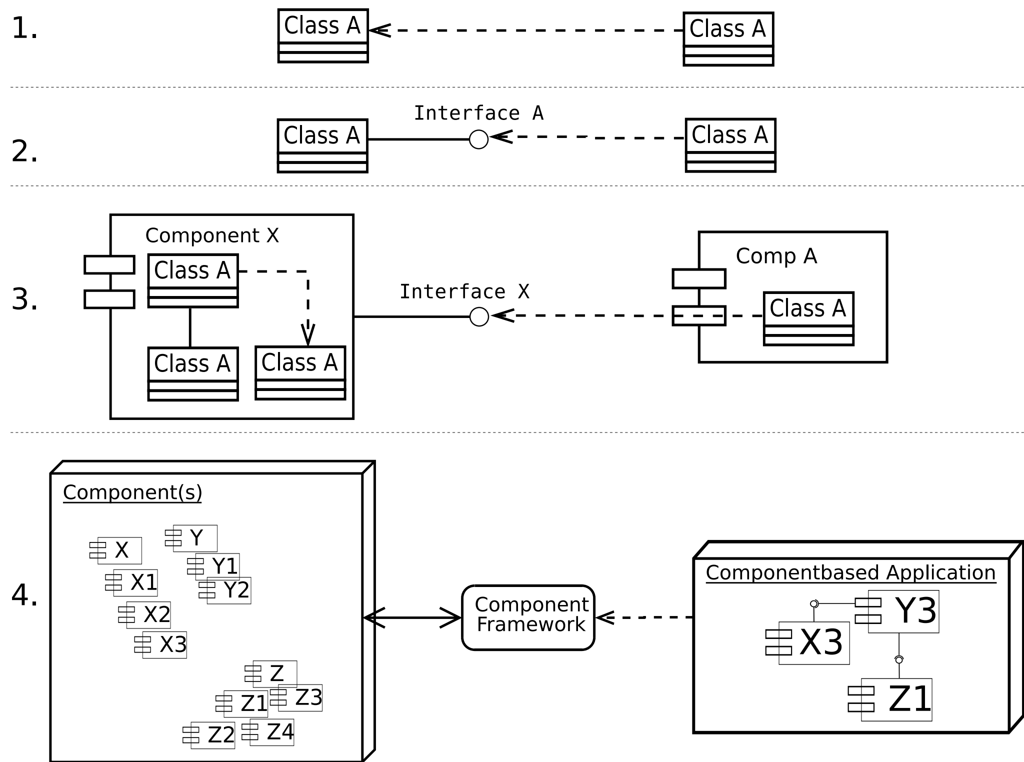

Abbildung 2.7: 1. Direkte Instanziierung und Methodenaufruf; 2. Trennung von Implementierung und öffentlichem Vertrag; 3. Interoperabilität, die Komponenten müssen nicht auf gleiche Systembedingungen aufbauen für die Verwendung; 4. Kompositionsanwendung - eine Kollektion interagierender und substituierbarer Funktionskomponenten (zum Bsp. X,X1,X2), die eine feststehende Schnittstellenvereinbarung miteinander eingehen

Dieses Verhältnis gibt einen Anhaltspunkt zum Grad der Portabilität einer Software [Moo]:

$$\text{Grad der Portabilität} = 1 - \frac{Aufwand\ der\ Portierung}{Aufwand\ der\ Neuimplementierung}$$

Was das anbelangt, so erläutert Tanenbaum et al. in *Guidelines for Software Portability* [TKB78] den Begriff Portabilität wie folgt:

> "Portability is the measure of the ease with which a program can be transferred from one environment to another; if the effort required to move the program is much less than that required to implement it initially, *and* the effort is small in absolute sense, then that program is highly portable."

Portierung ist das Gegenstück zur Neuimplementierung; ersteres baut auf eine vorhandene Implementierung, letzteres auf eine vorhandene Spezifikation auf. Bei der Implementierung neuer Software kommen häufig Überlegungen zur Wiederverwendung vorhandener Softwarebausteine auf. Im Abschnitt 2.3 wird der Begriff der Softwarekomponente eingeführt, der für diese verbreitete Art der Wiederverwendung prädestiniert ist. Die Portierung bezeichnet eine weitere Variante der Wiederverwendung, in diesem Sinne werden ganze Softwareprodukte auf neuen Plattformen wiederverwendet. Wie bei den Vorüberlegungen zum komponentenorientierten Design spielt die Wiederverwendung eine entscheidende Rolle. Bei neuen Softwareprojekten, ist die Entscheidung über ein portables Softwaredesign zu treffen, in anderer Weise muss für bestehende Software, sogenannter *legacy code*, über eine Portierung entschieden werden. Beide Varianten setzen auf die Quellcodeportabilität, die signifikant für den Entwicklungsprozess ist. Als weitere Möglichkeit existiert die, weit weniger beachtete Binärportabilität. Diese ist für ein portables Softwaredesign weniger von Bedeutung und wird hier nicht weiter betrachtet.

Dem Entwurf einer Software soll immer eine Anforderungsanalyse vorausgehen. Diese sollte neben der herkömmlichen Beleuchtung der Anwendungsaspekte auch die Schwerpunkte der Software im Hinblick auf die Portabilität betrachten und klar definieren. Der portable Softwareentwurf erfordert eine weitaus strengere Sicht auf die Mittel und Methoden, mit denen eine Software bewerkstelligt werden soll. Dazu hat die Arbeitsgruppe um James D. Mooney [Moo] drei grundlegende Prinzipien herausgearbeitet. Frei übersetzt, sind diese Aspekte für jeden Entwickler in diesem Bereich wichtig:

- kontrolliere die Schnittstellen
- isoliere Abhängigkeiten
- denke portabel

Die ersten beiden beziehen sich ganz konkret auf Aspekte des Entwurfs und der Implementierung. Der letzte Gedanke ist dem Durchhaltevermögen und der Disziplin der Entwickler geschuldet.

Grundlegend setzt sich ein Softwareprodukt aus mehreren, interagierenden Softwareelementen zusammen, die über definierte, interne Schnittstellen miteinander in Beziehung

stehen. Die internen Schnittstellen sollten, im Hinblick auf die Portabilität, bereits soweit abstrahiert werden, dass Abhängigkeiten und Annahmen, zu plattform- und implementierungsspezifischen Eigenschaften nicht nötig sind. Zusätzlich zu diesen internen Schnittstellen, werden externe Schnittstellen von der Betriebssystemplattform bereitgestellt. Diese ermöglichen systemnahe Zugriffe auf Hardwarebestandteile, Dateisysteme sowie spezifische Ein- und Ausgabegeräte. Unter Verwendung externer Schnittstellen, entsteht eine starke Abhängigkeit zur Betriebssystemumgebung. Externe Schnittstellen werden häufig verteilt über viele Softwareelemente verwendet. Diese sind verankert in der Funktionalität von Klassen und Methoden, was die systemnahe Kopplung dieser Einheiten entsprechend verstärkt. Unter Beachtung der Portabilität sollen jedoch lose gekoppelte Softwareprodukte entstehen, ohne spezifische Abhängigkeiten zur Plattform. Aus diesem Grund müssen vor dem Beginn der Implementierung klare Strukturierungen im Softwaremodell die systemabhängigen von den systemunabhängigen Bereichen kenntlich machen. Die weiteren Planungen zielen dann darauf ab, die systemspezifischen Abhängigkeiten zu minimieren. Für diese Aufgabe stehen zwei Methoden zur Verfügung:

- Isolierung
  Meint die Kapselung plattformspezifischer Funktionalität, in so wenig Modulen wie möglich. Dazu wird die Verwendung aller externen Schnittstellen gebündelt und über generische Zugriffsmethoden bereitgestellt.
- Standardisierung
  Bezeichnet die Verwendung plattformunabhängiger Systemaufrufe, zum Zweck einer durchgängig portablen Implementation.

Von diesen beiden Möglichkeiten ist bei der Entwicklung einer portablen Software, die Standardisierung wichtiger als die Isolierung externer Schnittstellen. Mit der ausgedehnten Betrachtung und Forschung im Bereich der Softwareentwicklung, der plattformspezifischen Merkmale und ihrer Gemeinsamkeiten haben sich mit der Zeit einheitliche Methoden und Mittel etabliert, die wesentlich für die Durchsetzung und Umsetzung eines portablen Softwareentwurfs sind. Beispielhaft sind hier POSIX konforme Systemaufrufe, ASCII basierte Zeichenkodierungen und IEEE normierte Zahlenbereiche zu nennen. Das führt den Gedanken weiter zur Implementierung und der nun notwendigen Selbstdisziplin der Programmierer bei der Umsetzung des Softwaremodells. Resultierend aus dem Designprozess sollte eine aussagekräftige Softwarespezifikation entstanden sein, die klar gekennzeichnete Schwerpunkte auf den gewünschten Grad an Portabilität setzt und diesen mit den geforderten Softwarebedürfnissen vereint. Der darauf folgende Implementierungsprozess greift sowohl für die Portierung vorhandener, alter Software *(legacy code)*, als auch für den von grundauf portabel modellierten Entwicklungsansatz. Zu Beginn der Implementierung ist die Wahl der passenden Programmiersprache eine grundlegende Entscheidung. Bei der Fülle an verfügbaren Sprachen reduziert sich die Auswahl mit Rücksicht auf die oben genannten Anforderungen auf einige wenige, die aufgrund ihrer Modellierung und standardisierter Plattformzugriffe (POSIX) in Frage kommen. Diese *Hochsprachen* abstrahieren die Interaktion mit dem zugrunde liegenden Betriebssystem und bieten plattformunabhängige Schnittstellen zur Verwendung an. Zu nennen sind hier exemplarisch Ada, C, C++ und Java, sowie C#. Angesichts der unterschiedlichen programmiersprachlichen Merkmale, sollten bei der Implementierung der Spezifikationen

keine Annahmen zu sprachspezifischen Eigenschaften getroffen werden. Damit besteht allerdings weiterhin ein begrenzter Anwendungshorizont, umrahmt von der eingesetzten Programmiersprache, denn nicht jede portierbare Programmiersprache ist geeignet für alle Arten von Anwendungsbereichen. Einen Ausweg wollen Forschungen zum Thema Code-Generierung aufzeigen [Fra94]. Dabei handelt es sich um die dynamische oder statische Generierung sprachspezifischer Ausprägungen aus einer Softwarespezifikation. Zur Laufzeit des Programms sollen dynamisch, benötigte Bestandteile generiert und in ausführbare Maschinenbefehle überführt werden. In anderer Weise meint eine statische Generierung, die Umsetzung der Softwarespezifikation unabhängig von einer Programmiersprache. Diese Zwischenstufe wird bei der Generierung als Vorlage genommen, um Quellcode für spezifische Programmiersprachen zu erzeugen. Diese Methode trägt zu einer sprachübergreifenden Quellcodeportabilität bei.

Eng verknüpft mit dem Entwurf und der Implementierung, ist eine durchdachte Dokumentation der Vorgehensweisen, Schwachstellen und Besonderheiten. Der Entwurf einer portablen Software erfordert hier zusätzlich die ausführliche Markierung kritischer Bereiche. Dazu zählen unumgänglich angenommene Systembedingungen, bestimmte Zugriffsmechanismen eigen implementierter Funktionen und natürlich Hinweise über die Betriebsfähigkeit der Software. Für eine spätere Portierung ist die Dokumentation der genannten Aspekte essentiell. Schließlich sollten portable Softwareprodukte nicht nur plattformübergreifend einsatzfähig sein und entsprechend erweitert werden können. Sie sind durch die Dokumentation geradezu verpflichtet, vor allem "entwicklerübergreifend" von den Menschen verstanden zu werden, die zukünftig die Wartung und/oder Portierung übernehmen.

Wie beim Komponentenparadigma sind mit der Portabilität Anforderungen und Kosten verbunden. Die Entscheidung für ein portables Softwaredesign wird häufig durch eine Kostenabschätzung bestimmt. In dieser Hinsicht definieren sich Kosten durch die Aufwendungen für Entwurf, Implementierung, zusätzlicher Wartung sowie die Extension eines Softwareproduktes während seines kompletten Lebenszyklus. Aus ökonomischer Sicht erfordert diese Entscheidung zu Beginn erhebliche Mehrinvestitionen im Entwurf und der Implementierung, rentiert sich jedoch im Nachhinein mit der Forderung nach einer Unterstützung zusätzlicher Plattformen. Ist die Dokumentation entsprechend ausführlich, lässt sich die Portierung in den normalen Implementierungsprozess einbetten. Damit würde, ausgehend von einer standardisierten Kernapplikation, eine Portierung nur die Ausgestaltung plattformspezifischer Module für externe Schnittstellen bedeuten. Durch den Einsatz eines portablen Softwaredesigns ergeben sich Programme, deren Transport zu anderen Zielplattformen und Adaptierung auf diese Plattformen sich wesentlich einfacher gestalten lassen. Softwaretechnisch ergeben sich weniger optimierte Programme, die allerdings flexibel in der Wahl der Systemplattform sind. Im Umkehrschluss sind Anwendungen, die auf eine spezifische Plattform zugeschnitten werden, entsprechend optimiert für die externen Schnittstellen der Plattform. Die Entwicklung ist kostengünstiger und bedarf nicht der Beachtung bestimmter Prinzipien der Portabilität. Wird jedoch der komplette Lebenszyklus einer Software ins Auge gefasst, so stellt sich bei der Erweiterung sowie der nachträglichen Portierung, ein finanziell erheblich höherer Mehraufwand ein, als es im Vorfeld des portablen Softwaredesigns der Fall ist. [Wik08d][TKB78][Ste78]

3 Das Plagiatanalysesystem Picapica

Die Plagiatanalyse ist generell gekennzeichnet durch die Frage, ob ein Autor in seiner Arbeit Teile von anderen Werken verwendet hat, ohne eine explizite Auszeichnung bzw. Zitierung. Damit muss geklärt werden, ob das Dokument des Autors plagiierte Abschnitte beinhaltet. Diese Frage lässt sich nur beantworten, indem alle bekannten Dokumente, stückweise mit dem des Autors verglichen werden. Bei diesem Problem stößt eine manuelle Bearbeitung schnell an ihre Grenzen, denn bereits der paarweise Vergleich aller Abschnitten zweier Dokumente ist sehr zeitaufwendig. Eine Betrachtung aller Dokumente, ist auf diese Weise unmöglich.

Picapica ist ein Softwareprodukt zur Analyse von Textdokumenten auf Plagiatsfälle. Über ein mehrstufiges Verarbeitungsmodell werden für benutzerdefinierte Dokumente Referenzquellen gesucht, die für die Dokumentinhalte als potentielle Plagiatquellen infrage kommen. Die vollautomatische Plagiatanalyse wird als Dienst über eine webbasierte Benutzeroberfläche zur Verfügung gestellt. Zum Einsatz kommen aktuelle Web-Technologien zur Visualisierung und Interaktion mit dem Softwaresystem. Der Dienst wurde in Java programmiert und ist daher plattformunabhängig. Das komponentenorientierte Design der Software, wird durch Javas ausgeprägtes Schnittstellenkonzept und seine Objektorientierung gestützt.

3.1 Prozessmodell der Plagiatanalyse

Die Plagiatanalyse von Picapica ist ein automatisierter Prozess zur Untersuchung von Dokumenten auf Plagiatverdachtsfälle. Die Entwickler hinter Picapica haben es sich zur Aufgabe gemacht, die menschliche Erkennensleistung von Plagiaten durch eine Software nachzubilden. Menschen haben den Vorteil, dass sie aus dem Kontext heraus die Semantik eines Sachverhaltes erschließen können. Im Sinne der Plagiatanalyse, handelt es sich um die Erkennung eines plagiierten Abschnitts, im Dokument eines Authors. Um das Plagiat als solches zu prüfen, muss die entsprechende Quelle ausfindig gemacht werden. Einem Softwaresystem bleiben die semantischen Hürden nicht erspart, so wird die Software nicht verstehen, was sie zu welchem Zweck sucht und vergleicht. In anderer Weise ist Software in der Lage, mit großen Dokumentmengen umzugehen und darauf Algorithmen anzuwenden. Diese Möglichkeit ist durch eine manuelle Verarbeitung nicht mehr gegeben. Die Entwickler von Picapica haben das Problemfeld der Plagiatanalyse erfasst und in kleine Teilprobleme zerlegt, die sie einfacher durch ein Softwaresystem zu lösen versuchen [BSP07].

3.1.1 Kernaufgaben der Plagiatanalyse

Die Plagiatanalyse bezeichnet einen Prozess zur Plagiaterkennung. Dieser kann in verschiedenen Softwaresystemen unterschiedlich ausgeprägt sein. Bei Picapica wurde ein dreistufiger Prozess zur Plagiatanalyse erdacht, bestehend aus

- heuristischem Retrieval,
- detailierter Analyse und
- Nachverarbeitung.

In dieser Reihenfolge findet die Verarbeitung eines verdächtigen Dokuments statt (Abbildung 3.1).

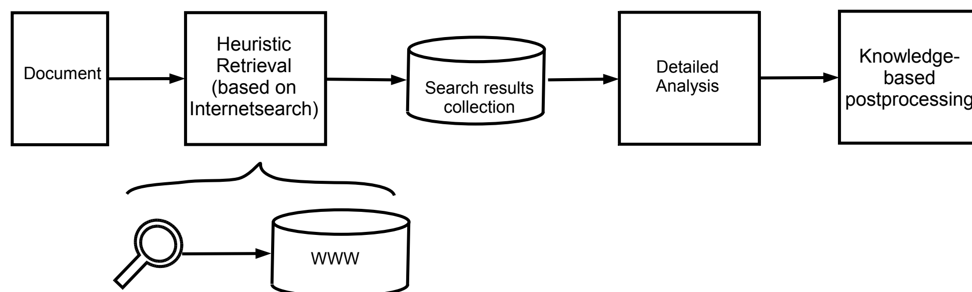

Abbildung 3.1: Der Verarbeitungsprozess der Plagiatanalyse als Pipeline.

3.1.2 heuristisches Retrieval

Das heuristische Retrieval benutzt markante Textbausteine eines Verdächtigen Dokuments, zur Abfrage eines speziellen Dokumentindex. Dieser Dokumentindex hält als Schlüssel eine Reihe dieser Textbausteine und verknüpft diese mit realen Dokumenten einer Kollektion. Die Textbausteine stellen einen Suchbegriff aus Schlüsselwörtern dar, der bei Picapica zur Abfrage der Internetsuchmachinen Google, Yahoo und MSN genutzt wird Die Internetsuchmaschinen liefern potentielle Referenzdokumente als Ergebnis der Abfrage. Diese Kandidatendokumente sind eine Vorauswahl aller indizierten Dokumente der Suchmaschinen. Die Vorauswahl verringert die Anzahl der anschließend zu vergleichenden Dokumente.

3.1.3 detailierte Analyse

Unter einem stärkeren Retrieval-Modell als dem heuristischen, findet eine tiefergehende Analyse der Dokumente statt. Zur Anwendung kommen Algorithmen, die die häufigsten Methoden des Plagiarismus erkennen:

- *identische Kopie*
  Das Quelldokument ist die Kopie eines existierenden Dokuments.
- *stückweise Kopie*
  Abschnitte des Quelldokuments wurden aus einem Referenzdokument kopiert.
- *modifizierte Kopie*
  Das Quelldokument weist modifizierte Abschnitte eines Referenzdokuments auf.
- *Stilbrüche*
  Der Schreibstil des Autors hat sich untypisch, über das Dokument hinweg, geändert.

Dabei werden immer paarweise, alle gefundenen Referenzdokumente, mit dem verdächtigen Quelldokument verglichen, um eine Aussage über ihre Ähnlichkeit zu treffen. Eine Methode ganze Dokumente gegeneinander in ihrer Ähnlichkeit abzuschätzen, ist die Kosinusähnlichkeit. Dafür notwendig ist ein starkes Retrieval-Modell, wie das Vector-Space-Model. Hier werden Dokumente als hochdimensionale Vektoren representiert. Der Vektor ist damit so lang, wie das Dokument Wörter beinhaltet. Durch diese Darstellung beschreibt die Kosinusähnlichkeit den Kosinus des Winkels, zwischen zwei zu vergleichenden Dokumentvektoren. Umso näher dieser Wert gegen Eins geht, desto ähnlicher sind sich die Dokumente. Damit wird ein weitere Vorauswahl geschaffen, um Referenzdokumente auszusondern, die unterhalb eines bestimmten Schwellwerts der Ähnlichkeit liegen. Referenzdokumente mit einer hohen Ähnlichkeit zu dem verdächtigen Dokument, werden in dieser Hinsicht als identisch klassifiziert, dabei wird durch einen Schwellwert bestimmt, welche Referenzdokumente auf diese Heuristik zutreffen. Die übriggebliebenen Dokumente der Vorauswahl, die nicht als identisch klassifiziert wurden, werden anschließend durch speziellere Algorithmen einem Vergleich unterzogen. Jedes Dokumentpaar wird abschnittsweise durchlaufen und jeder Abschnitt des Quelldokuments mit allen anderen, des gefundenen Referenzdokuments verglichen. Das kann trotz der Vorauswahl zu entsprechend komplexen Operationen und langen Laufzeiten führen, wenn die Dokumente sehr lang sind. Die Auswahl optimaler Algorithmen ist für diesen Einsatz unumgänglich. Diese bauen vorwiegend auf Hashing-basierte Verfahren auf und nutzen auftretende Hashkollisionen als Ähnlichkeitsindikator [SP06]. Eine kryptographische Analyse berechnet für alle möglichen Abschnitte der beiden Dokumente, einen eindeutigen Hashwert. Taucht der gleiche, eindeutige Hash in beiden Vergleichsdokumenten auf, dann wurde dieser Abschnitt unverändert kopiert. Das spezialisierte *Fuzzy-Fingerprinting* [Ste05][SM07], ermöglicht über eine unscharfe Analyse der Dokumente, eine Aussage über deren Ähnlichkeit. Dabei werden die hochdimensionalen Dokumentvektoren auf einen niedrigdimensionalen Raum reduziert, bei dem jede Dimension die Häufigkeit der Worte mit gleichem Anfangsbuchstaben misst. Nach dem latainischen Alphabet besteht dieser Vektorraum aus 26 Dimensionen. Anschließend wird die Abweichung der gemessenen Häufigkeit zum Erwartungswert jeder Dimension des Dokumentvektors berechnet. Besitzen zwei Dokumentvektoren die gleiche Abweichung in der gleichen Dimension, ist das ein Indiz für die Ähnlchkeit. Um den Hashwert eines Dokuments zu berechnen, werden die Abweichungen jeder Dimension über ein Fuzzyfizierungsschema auf ganzzahlige Werte abgebildet. Dieses Schema ordnet definierten Intervallen ganzzahlige Werte zu, diese Werte aufsummiert ergeben den Hashwert. Durch dieses Verfahren können auch für abschnittsweise modifizierte Dokumente, die

gleichen Hashwerte errechnet werden. Die Kollision dieser unscharfen (fuzzy) Hashwerte, ist ein Indiz dafür, dass Wörter verschoben oder Zeilen vertauscht wurden der Text demnach umformuliert, der beabsichtigte Zusammenhang jedoch nicht geändert wurde. Ein weiteres Verfahren zur Plagiatanalyse, das vollständig ohne den Vergleich zu einem Referenzdokument auskommt, ist die intrinsische Analyse zur Plagiaterkennung [MS06]. Vereinfacht gesagt behandelt dieses Verfahren Abweichung in der Kontinuität des Quelldokuments. Damit ist die Stetigkeit des Schreibstils eines Authors gemeint, der sich im Allgemeinen während eines Textes nicht ändert. Treten Abweichungen auf, so kann hier eine externe Quelle vermutet werden, die in die Formulierungen eingeflossen ist.

3.1.4 Nachbearbeitung

Die Nachbearbeitung ist der letzte Schritt im Analyseprozess. Aufgrund der Tatsache, dass die eingesetzten Verfahren sich in ihren Ergebnissen teilweise überlappen können, ist die Forderung nach einer einheitlichen und vor allem organisierten Ergebnisstruktur zu berücksichtigen. In dieser Hinsicht werden die Ergebnisse der unterschiedlichen Algorithmen, normalisiert und vereinheitlicht. Die Ergebnisbereiche werden wiederholt gegeneinander verglichen und anschließend gegebenenfalls vereinigt. Dadurch können mögliche *false positives*, also fälschlicherweise identifizierte Abschnitte, herausgefiltert werden. Diese Ergebnisse sind nicht auf algorithmische Fehler zurückzuführen, sondern bedingt durch den Einsatz statistischer Verfahren in den Analysen einzukalkulieren. Während der Nachbearbeitung wird dieser Tatsache Rechnung getragen und entsprechend entgegengewirkt. Die anschließende Serialisierung der Ergebnisdaten erzeugt die finalen Ausgabedateien für die Web-Oberfläche.

3.2 Ausführungsmodell und Server-Architektur

Im zweiten Kapitel wurden bereits grundlegende Paradigmen der Softwareentwicklung aufgeführt, die im Hinblick auf das Plagiatanalysesystem Picapica von Bedeutung sind. Aus der Anforderungsanalyse, die im Vorfeld des Systementwurfs stattfand, wurden folgende Aspekte identifiziert, die das Softwaresystem umsetzen sollte.

- Das Analysesystem soll auf einem Rechnerverbund lauffähig sein, bei dem jeder Rechner alle Teilaufgaben der Plagiatanalyse ausführen können soll. Die notwendigen Daten für die Analyseaufgaben, sind von jedem Rechner selbst zu verwalten, um eine dezentrale Datenhaltung umzusetzen.
- Das Analysesystem soll sich an neue Erkenntnisse in der Forschung zur Plagiatanalyse anpassen lassen. Aus diesem Grund wird eine dynamische Zusammensetzung von Teilaufgaben gefordert, die über einheitlich definierte Schnittstellen in das System integriert werden. Die Teilaufgaben der Plagiatanalyse sollen sich ändern, modifizieren, erweitern und neu kombinieren lassen. Diese Möglichkeiten bietet die komponentenorientierte Softwareentwicklung.
- Die nötige Kommunikation unter mehreren, verteilten Rechnern soll über eine gesonderte Einheit möglich sein, die zentral alle nötigen Informationen einer Analyse

hält. Der direkte Datenaustausch als auch die Kommunikation sollen über einfache Standardprotokolle erfolgen.

- Das Softwaresystem soll robust und skalierbar sein. Damit ist die Ausfallsicherheit und Fehlertoleranz gemeint. Zeitweise Ausfälle oder Fehlfunktionen müssen durch andere Rechner weitgehend kompensiert werden. Durch optimale Ressourcenausnutzung und freier Zusammensetzung der Rechner soll der Durchsatz des Gesamtsystems verbessert werden.

Picapica ist ein Softwaresystem, dass über mehrere Versionen zu dem geworden ist, was diese Ausarbeitung erläutert. Damit ist aus den Anforderung mit der Zeit eine Software erwachsen, die unterschiedliche Prinzipien der Softwareentwicklung verinnerlicht und intensiv anwendet.

Picapica setzt auf ein Netzwerk von Computersystemen, die über eine Middleware miteinander in Beziehung stehen. Diese verteilte Softwarearchitektur (Abbildung 3.2) besteht aus mehreren Servern, die in folgenden Bereichen des Softwaresystems eingesetzt werden:

- Analyse
- Kommunikation und Datenaustausch
- Benutzerinteraktion

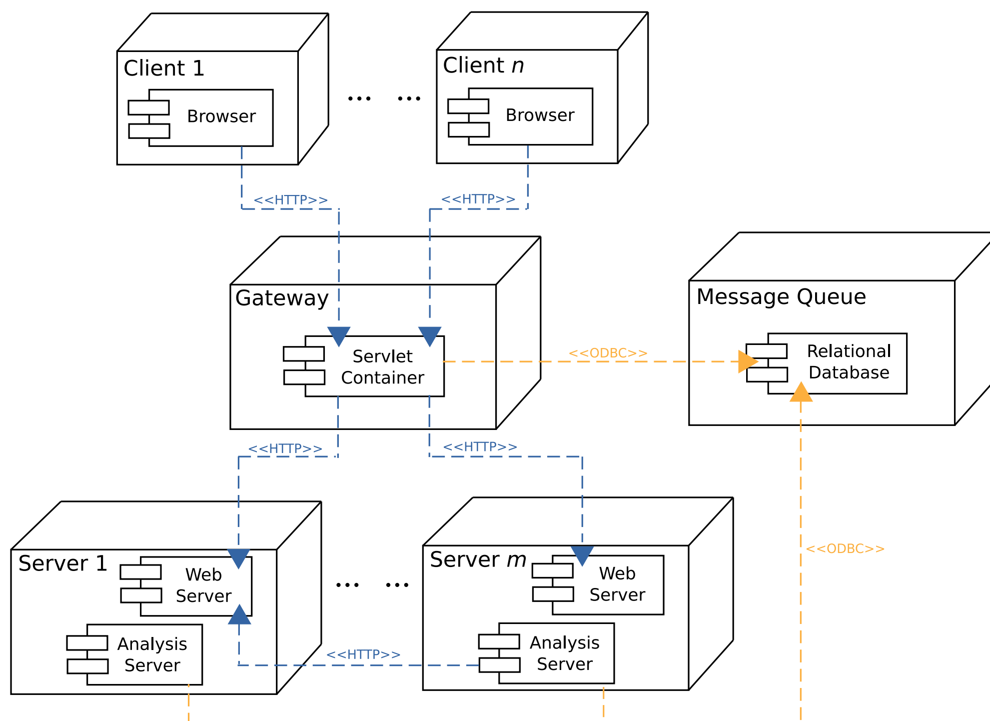

Abbildung 3.2: Übersicht der Serverkonstellationen im Systemmodell von Picapica. Der Nachrichtenaustausch mit dem Datenbankserver erfolgt mit Hilfe der ODBC-Technologie. Der Datenaustausch unter allen anderen beteiligten Servern und Clients findet über das HTTP-Protokoll statt.

3.2.1 Analyse

Die Analysesoftware jedes Rechners wird als Jobserver bezeichnet. Als Rahmenprogramm zur Bearbeitung einzelner Analyseaufgaben setzt der Jobserver ein jobbasiertes Verarbeitungsmodell ein. Eine Analyseaufgabe wird als Job bezeichnet und kann auf jedem Jobserver ausgeführt werden, falls alle notwendigen Vorbedingungen erfüllt sind, wie zum Beispiel die Ergebnisse logisch vorher auszuführender Jobs. Über eine einheitliche Schnittstelle ist jeder Job in das System integriert und stellt seine Funktionalität der Middleware zur Verfügung. Bei Picapica ist ein Job als funktionaler Container zu sehen, er kombiniert bereits ausgereifte Funktionalität aus vorhandenen Bibliotheken. Die einheitliche Schnittstelle der Jobs, lässt die Kombination alter und neuer Funktionalität im Gesamtsystem zu, womit die Anpassung des Systems an neue Forschungserkenntnisse erleichtert wird. Das flexible Verarbeitungsmodell realisiert die weitgehend losgelöste Behandlung der Analyseaufgaben durch mehrere Jobserver. Parallel laufende Jobserver erhöhen den Durchsatz des Gesamtsystems, indem jeder Jobserver seine verfügbaren Kapazitäten bereitstellt. Gleichzeitig können Ausfälle durch andere Jobserver kompensiert werden, was die Stabilität der Verarbeitung fördert. Entscheidend für die Vollständigkeit der Plagiatanalyse ist ein geordneter Ablauf logisch zusammenhängender Jobs und die Verfügbarkeit ihrer Ergebnisse. Dafür ist eine intensive Kommunikation unter den Jobservern nötig, die durch das Konzept der *Message-Oriented-Middleware* umgesetzt wird. Es beinhaltet den indirekten Nachrichtenaustausch zwischen den Jobservern über eine externe Nachrichtenquelle – die *Message-Queue*.

3.2.2 Kommunikation und Datenaustausch

Bei Picapica wird die Message-Queue durch einen *MySQL Datenbankserver* umgesetzt, der in einer Datenbank alle notwendigen Informationen der Jobs hält. Der Nachrichtenaustausch erfolgt über eine aktiven Verbindung der Jobserver zur Message-Queue, unter Verwendung einer standardisierten Verbindungstechnologie – der ODBC *(Open Database Connectivity)*. Eine Nachricht setzt sich zusammen aus der Jobbezeichnung, zugehörigen Parametern und Metainformationen. Jeder Jobserver erfragt in regelmäßigen Intervallen von der Message-Queue neue Nachrichten und aktiviert im Rahmen seiner freien Ressourcen die entsprechenden Jobs. Alle daraus folgenden Jobs werden wieder als Nachricht der globalen Message-Queue hinzugefügt. Die Ein- und Ausgabedaten jedes Jobs werden über die Message-Queue verwaltet und sind in den Parametern kodiert. Dafür wurde ein generisches Datenmodell, als Bedingung für jeden Job vereinbart:

- Jeder Job erhält als Eingabe eine oder mehrere Ressourcenadressen. Diese URIs *(Universal Resource Identifier)* verweisen auf die entsprechenden Eingabedaten und werden vom Job zur Verarbeitung heruntergeladen.
- Nach der Verarbeitung erzeugt jeder Job für seine Ergebnisse ebenfalls URIs als Ausgabe.
- Diese URIs sind Bestandteil der Parameter eines möglichen Nachfolgejobs, die als Nachricht von der globalen Message-Queue kommen. Damit definiert jeder Job durch seine Ausgabe, die Eingabe seines Nachfolgers.

Die URIs zu jedem Job, verweisen auf den Ablageort der Ein- und Ausgabedateien und werden in Verbindung mit dem standardisierten *Hypertext Transfer Protocol (HTTP)* dazu benutzt, diese Dateien von einem Web-Server abzurufen. Jedem Jobserver ist ein Web-Server zur Seite gestellt, der die lokale Datenhaltung übernimmt. Aufgrund der nachrichtenorientierten Kommunikation kann die Anzahl der Rechner mit Jobserver und Web-Server dynamisch variiert werden. Damit können die verlinkten Ein- und Ausgabedateien der Jobs, während einer Analyse auf unterschiedlichen Rechnern gehalten werden. Der notwendig Download durch die Jobs erfolgt wie bei einem herkömmlichen Web-Browser.

Zusammenfassend existieren bei Picapica zwei Kommunikationswege unter den Jobservern:

- der direkte Download der Daten von einem Web-Server, der jedem Jobserver beigestellt ist und
- der indirekte Nachrichtenaustausch über die Message-Queue, der essentiell für das Softwaresystem ist.

Die Message-Queue hält global alle Nachrichten der Jobserver. Bei Picapica sind das die zur Bearbeitung offenstehenden Jobs, ihre Ein- und Ausgaben. Die Adressen dazu stehen als URIs kodiert in der Message-Queue. Damit ist sie eine zentrale Instanz dieses Softwaresystems. Hier sind alle Informationen abgelegt, die für die Koordination der Jobs und den Datentransfer nötig sind. Entsprechend stabil und ausfallsicher ist diese Einheit durch den MySQL-Server realisiert. Die Message-Queue sorgt für die Kommunikation unter den Servern von Picapica, und so wird sie auch bei der Oberfläche eingesetzt, denn die Plagiatanalyse wird durch eine Nachricht der Oberfläche gestartet. Diese web-basierte Oberfläche soll nachfolgend, gesondert behandelt werden.

3.2.3 Benutzerinteraktion

Picapica ist konzipiert als web-basiertes Informationssystem und mit einem Web-Dienst vergleichbar. Eine web-basierte Oberfläche stellt die benutzerfreundliche Schnittstelle zum Softwaresystem dar. Moderne Web-Technologien werden sowohl auf Anwenderseite als auch auf Serverseite eingesetzt. Die Benutzeroberfläche ist aufgrund der eingesetzten Java-Servlet Technologie zweigeteilt. Ein *Servlet-Container* publiziert die Web-Oberfläche für den Web-Browser und stellt gleichzeitig die funktionale Anbindung zum Analysesystem bereit. Damit ist die Oberfläche gegliedert in Frontend auf Anwenderseite und Backend auf Serverseite. Auf Serverseite werden Java-Servlets eingesetzt, die unter anderem die Verbindung der Web-Oberfläche zum Analysesystem bereitstellen. Diese Verbindung kommt wie bei den Jobservern über ODBC zustande. Das Backend wertet die Nachrichten der Message-Queue aus und übermittelt sie als Status- und Ergebnisinformationen dem Frontend.

3.3 Web-basierte Visualisierung von Analyseergebnissen.

Zur Visualisierung der Oberfläche und der Analyseergebnisse im Web-Browser, kommen HTML *(Hypertext Markup Language)* und CSS *(Cascading Stylesheet)*, als Auszeichnungssprachen für Inhalt und Layout zur Anwendung. Das Frontend beinhaltet die clientseitige, Javascript basierte Logik für die Kommunikation mit dem Backend und die Interaktion mit dem Benutzer. Zur Kommunikation wird *Ajax*, die asynchrone Kommunikationsfähigkeit der Scriptsprache Javascript verwendet. Die Analyseergebnisse holt sich das Frontend über HTTP von den Web-Servern zu jedem Jobserver. Die Verwendung möglichst grundlegender Definitionen im Bereich Javascript, HTML und CSS erlaubt die browserübergreifende *(cross-browser)* Verwendung der Oberfläche in den gängigen Web-Browsern.

Die Web-Oberfläche bietet dem Anwender die Möglichkeit, eigene Textdokumente hochzuladen und dafür eine Plagiatanalyse zu starten (Abbildung 3.3). Dabei spielt sowohl das Format, als auch der Ort des Dokuments eine untergeordnete Rolle. Über ein Formular lassen sich neben herkömmlichen Dateien, auch Web-Links und reiner Text hochladen. Zusätzliche existiert eine Auswahl, der zur Verfügung stehenden Methoden, die das System nutzen soll um geeignete Vergleichsreferenzen zu finden. Werden die Informationen vom Anwender hochgeladen wird für den Analysevorgang eine Session-Id registriert. Die Session-Id wird vom Backend generiert und bei jedem Aufruf durch das Frontend übermittelt. Solang ein neuer Benutzer keinen Upload und damit keine Analyse gestartet hat, ist die Session-Id nicht aktiv. Der Start einer Analyse erfolgt in Verbindung mit der Session-Id der aktuellen Sitzung. Ein Servlet registriert die neue Session (Sitzung) bei der globalen Message-Queue und sendet eine Nachricht für den ersten Job der Plagiatanalyse. Die im Hintergrund anfragenden Jobserver, beginnen mit ihrer Arbeit und schreiben Folgejobs in die Message-Queue. Während des gesamten Prozesses bekommt der Anwender den aktuellen Analysestatus angezeigt. Dieser beinhaltet, neben der Auflistung zwischenzeitlich gefundener Resultate, auch die abgeschlossenen und noch offenen Jobs. Die gefundenen Verdachtsfälle für Plagiate, sind für eine Ansicht innerhalb der Web-Oberfläche aufbereitet und ermöglichen eine vergleichende Ansicht des Quelldokuments und potentieller Plagiatquellen. Farbliche Markierungen heben die gefunden Plagiatstellen entsprechend ihrer Genauigkeit hervor. Anwendung findet hier je nach Ergebnistyp eine übersichtliche Zwei-Fenster-Technik (Abbildung 3.4) oder Ein-Fenster-Technik (Abbildung 3.5 und 3.6) , die ein direktes Anspringen der farbigen Plagiatstellen ermöglicht.

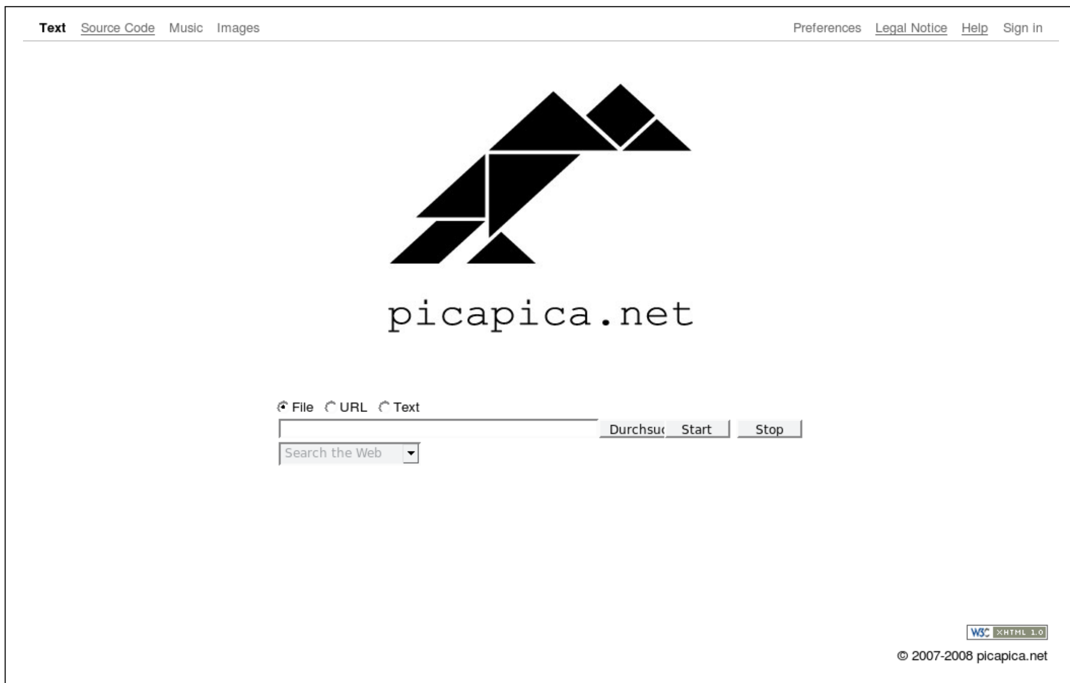

Abbildung 3.3: Startseite von Picapica.

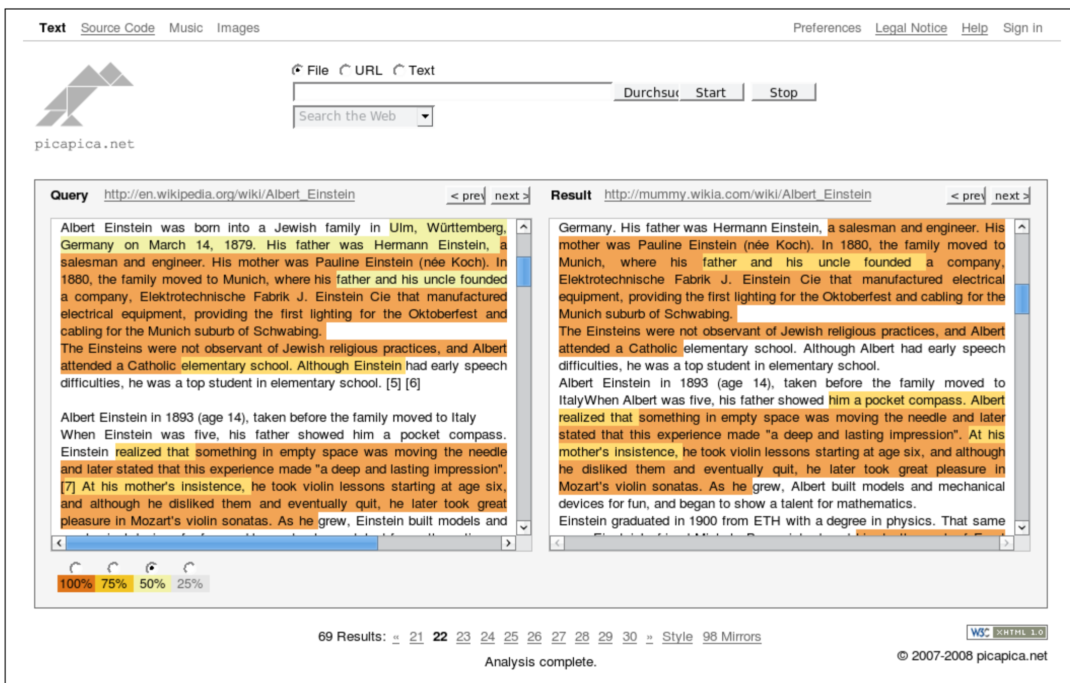

Abbildung 3.4: Analyseergebnisse der kryptographischen Analyse und des Fuzzy-Fingerprinting, ersichtlich durch unterschiedliche Farbtöne.

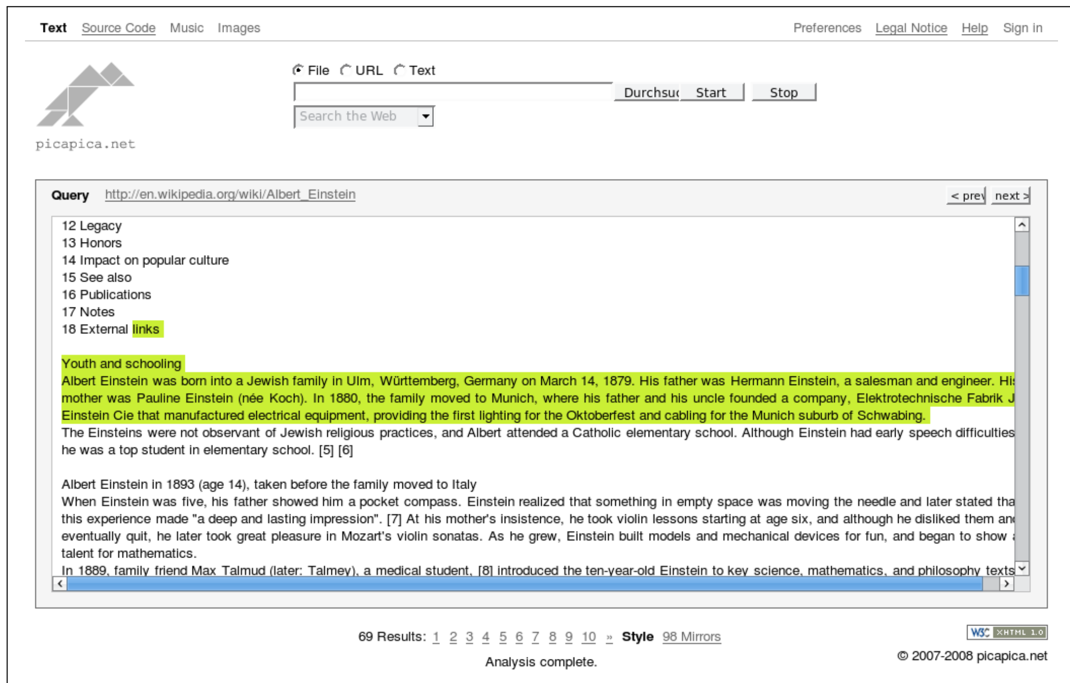

Abbildung 3.5: Analyseergebnisse der intrinsischen Analyse.

Abbildung 3.6: Auflistung der Referenzdokumente, die durch die Heuristik der Konsinusähnlichkeit als identische Kopien klassifiziert wurden.

3.4 Softwarekomponenten

Im Vorfeld der komponentenorientierten Entwicklung, wurden die Kernaufgaben der Plagiatanalyse identifiziert und in angemessene Komponenten gruppiert. Diese *Analysekomponenten* existieren neben zusätzlichen Komponenten für die Benutzerinteraktion und Grundlagentechnik. Die Identifikation, der zu kombinierender Funktionalität, ist ausschlaggebendes Kriterium für eine Komponente. Sie bestimmt ihre Granularität und spätere Einsatzmöglichkeiten. Die Frage der Granularität wurde bereits diskutiert und soll hier als Grundlage der Ausführung dienen.

3.4.1 Interne Komponenten

Jede identifizierbare Komponente des Picapica-Systems, ist für die Gesamtfunktionalität von Bedeutung. Es handelt sich also um passive Komponenten, die nur im Verbund verwendet werden können. Damit setzt sich das Softwaresystem zusammen aus:

- picapica-common
- picapica-heuristic-retrieval
- picapica-detailed-analysis
- picapica-postprocessing
- picapica-ui

Der für Picapica typische Dreischritt, heuristisches Retrieval, detaillierte Analyse und Nachbearbeitung, schlägt sich in der Komponentensicht nieder (Abbildung 3.7).

Aufgrund der intensiven Forschung im Bereich der Plagiatanalyse [BSP07] standen den Entwicklern erprobte Technologien zur Verfügung, die es entsprechend dem Prozessmodell zu kombinieren galt. In dieser Hinsicht wurden die Technologien aus den Bereichen Information-Retrieval und Plagiaterkennung in Bibliotheken organisiert. Genau diese Bibliotheken kommen in den Jobs und Zusatzfunktionalitäten des Plagiatanalysesystems zum Einsatz. Damit ist ein flexibler Technologietransfer geschaffen, der es den Entwicklern der Bibliotheken von Picapica erlaubt, ohne Abhängigkeiten getrennt voneinander die Software zu warten und weiterzuentwickeln. Die internen Softwarekomponenten von Picapica setzen sich verallgemeinert, wie folgt zusammen:

picapica-common
: In dieser Komponente ist die systembezogene Grundlagenfunktionalität integriert. Dazu zählt die Kommunikation mit der Message-Queue, die intern vom Jobserver für den Nachrichtenaustausch benötigt wird. Weiterhin die Datenformate für die jobspezifischen Ausgabedaten. Sie setzen sich aus einem intermediären XML-Format zusammen, das die Texte und plagiierten Stellen auszeichnet.

picapica-heuristic-retrieval
: Das heuristische Retrieval setzt sich aus Suchtechnologien zusammen, um aus indizierten Dokumentkollektionen Kandidatendokumente bzw. mögliche Referenzdokumente zu filtern. Eine Suchterm-Extrahierung liefert markante Textbausteine

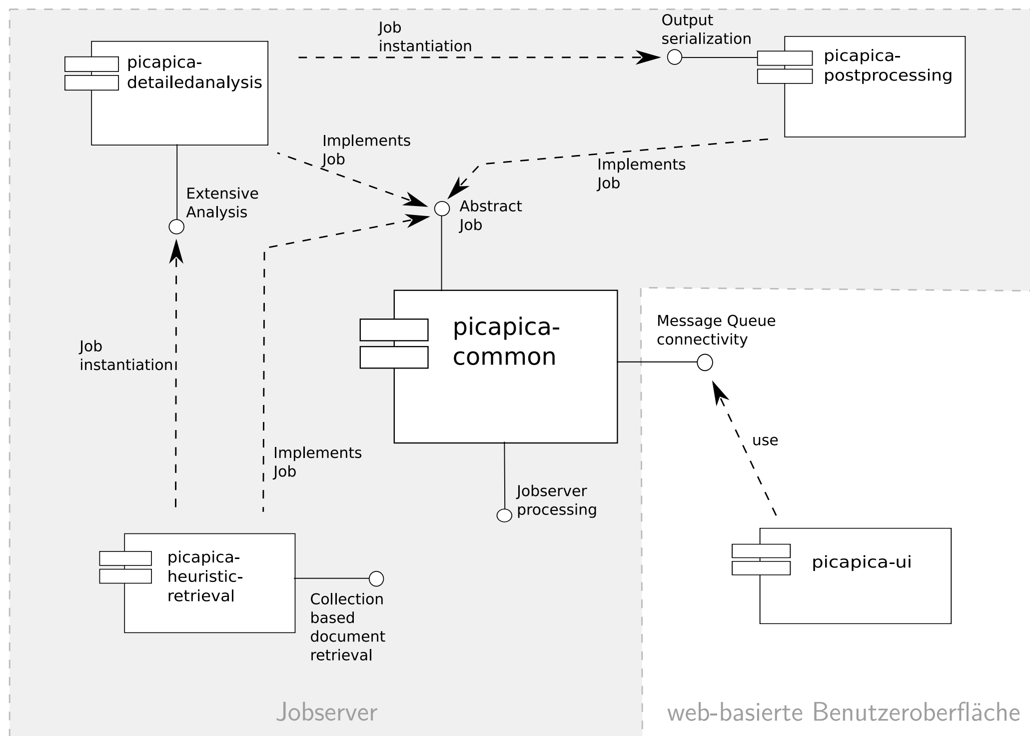

Abbildung 3.7: Übersicht der Komponentenkonstellationen von Picapica.

des Quelldokuments, die für die Anfrage an den den Dokumentenindex genutzt werden.

picapica-detailed-analysis
Diese Komponente ist algorithmisch gesehen die Kernkomponente des Systems. Spezialisierte Algorithmen für die Erkennung plagiatbehafteter Dokumente, vergleichen paarweise alle möglichen Referenzdokumente mit dem Quelldokument, abschnittsweise und vollständig. Die Komponente gruppiert die Verwendung des Fuzzy-Fingerprinting, der Kosinusähnlichkeit, der intrinsichen Analyse und der kryptographischen Analyse.

picapica-postprocessing
Einheitliche Ausgabeschnittstellen für Ergebnisdaten und deren nötige Nachbearbeitung, sind in dieser Komponente gruppiert.

picapica-ui
Diese Komponente beinhaltet die Benutzerschnittstelle für Interaktionen und Visualisierung der Analyse. Sie integriert die Implementierung der Java-Servlet basierten Web-Applikation und der Web-Oberfläche.

Der Jobserver als strukturelle Einheit setzt sich zusammen aus vier Komponenten, wie sie in Abbildung 3.7 grau hinterlegt sind. Jede dieser Komponenten ist als Kollektion von Jobs und zusätzlicher Funktionalität zu charakterisieren. Dabei nutzen die an der Analyse beteiligten Komponenten, eine von *picapica-common* verfügbare Jobschnittstelle. Diese Schnittstelle besitzt nur die Funktionalität, Informationen mit dem Rah-

menprogramm, Jobserver, auszutauschen. Jede beteiligte Komponente stellt Jobs für ihren Aufgabenbereich zur Verfügung. Diese sind Spezialisierungen der Jobschnittstelle, indem diese mit Funktionalität externer Bibliotheken ausgestattet wird. In dieser Hinsicht kombinieren Jobs etablierte Fähigkeiten, ohne direkte Abhängigkeiten dazu herzustellen. Damit lassen sich Jobs mit beliebiger Funktionalität füllen. Zusammenfassend sind diese Komponenten nicht nur funktionale Gruppierungen, sie spiegeln auch die Hauptarbeitsbereiche der Plagiatanalyse wieder.

Weiter oben wurde gesagt, dass das Softwaresystem aus dem Jobserver, samt zugehörigem Web-Server und einem Datenbankserver besteht. Im Hinblick auf die drei beteiligten Servertypen bei Picapica, wird eine weitere Facette der Komponentenorientierung von Picapica deutlich. Der Datenbankserver, der Jobserver und der damit kombinierte Web-Server lassen sich als funktionale Komponenten gliedern. Hinzu kommt bei dieser Sicht die Java-Servlet basierte Web-Oberfläche, als vierte funktionale Komponente des Plagiatanalysesystems.

Das komponentenorientierte Softwaredesign erschließt sich bei Picapica über dessen funktionale Gruppierung und Softwarestruktur. Die Java-Umgebung unterstützt dieses Softwaredesign, durch die paketbasierte Gruppierung der Funktionalität und durch das starke Schnittstellenkonzept.

3.4.2 Externe Komponenten

Für die korrekte Umsetzung, der in Abschnitt 3.1 erläuterten Kernaufgaben der Plagiatanalyse, sind zusätzliche Funktionalitäten unabdingbar. Im Folgenden sind das, die Konvertierung der Eingabedateien in ein einheitliches Format und der Download von Dateien zur Analyse. Für die Aufgabe der Konvertierung werden unterschiedliche Bibliotheken eingesetzt, die eine Vielzahl von Formaten unterstützen. Die Analyseverfahren verwenden als Eingabe reine Textdokumente ohne Formatierung. Diese stellen somit den kleinsten gemeinsamen Nenner aller bekannten Formate dar. Verwendet wird bei Picapica eine eigens entwickelte Bibliothek, die diese externe Bindung abstrahiert. Die nötigen Programme gehören zur minimalen Ausstattung eines Rechners zur Plagiatanalyse und sind wichtig für die Funktionalität der Jobs. Der Dateidownload wird in gleicher Weise über eine effiziente, externe Software bewerkstelligt, obwohl Java ebenfalls einen Dateidownload mit einer URI über HTTP unterstützt. Die externe, freie Software wird insbesondere von einer großen Community getragen, während projektinterne Entwicklungen proprietärer Natur waren.

Zusammenfassend verwendet das Picapica-System folgende, externe Anwendungen:

- OpenOffice
  Die freie Open-Source Office-Suite wird für zur Konvertierung etlicher Dokumentformate, wie zum Beispiel den Formaten von Microsoft Office, StarOffice und OpenOffice benutzt.
- Ghostgum und Ghostview
  Diese Anwendungen erlauben die Konvertierung von PDF-Dateien und PostScript-Dateien.

- Wget
  Der ursprünglich aus dem Unix-Umfeld stammende Downloader, ist für sämtliche Downloadaktivitäten des Plagiatanalysesystems verantwortlich.

Mit der Nutzung dieser externen Software erhöhten sich zwar die Abhängigkeiten zur Systemplattform, allerdings sind die eingesetzten Anwendungen durch die breite Verwendung, auf unterschiedliche Systeme portiert. Die infrage kommenden Systemplattformen, sind aus der Sicht der Entwickler:

- Microsoft's Windows Betriebssystem ab der Version – Windows XP
- "unixoide" Linux-Distributionen, wie Debian, SuSE u.a.
- Apple's Mac OS X

In Bezug auf die Web-Oberfläche, wird ebenfalls die Anzeige und Interaktion auf den gängigen Browsern dieser Betriebssysteme unterstützt, welche sich unter anderem folgendermaßen zusammensetzen:

- Internet Explorer ab Version 6
- Firefox ab Version 2.x
- Opera ab Version 9.x
- Safari ab Version 3.x mit Mac OS X "Leopard"

Diese externen Abhängigkeiten beziehen sich auf die unterstützende Software. Die Implementierungssprache von Picapica ist aufgrund ihrer Eigenschaften von vornherein für die genannten Systemplattformen geeignet. Zum Einsatz kommt hier die rein objektorientierte Programmiersprache Java. Die für Java typische *Virtual-Machine* ist eine entscheidende Technologie für die Plattformunabhängigkeit dieser Sprache. Durch den vom Java-Compiler erzeugten Bytecode, einer intermediären Ausführungsebene, kann dieser zu allen Plattformen transportiert werden für die eine entsprechende *Virtual-Machine* existiert. Diese nutzt einen Bytecode-Compiler zur Übersetzung des Bytecode in plattformspezifischen Maschinencode. Das Plagiatanalysesystem Picapica wird so unabhängig von der Entwicklungsplattform und den zukünftigen Laufzeitplattformen.

3.5 Portable Plagiatanalyse

Das portable Picapica versteht sich nicht als Neuentwicklung, sondern baut vielmehr auf denselben Prinzipien des Web-basierten Systems auf. Hinzu kommen einige spezielle Aspekte, die in diesem Abschnitt erläutert werde sollen.

Die Forderung nach einer portablen Version des Plagiatanalysesystems, resultiert aus der Idee, die Software an all diejenigen Benutzer zu vergeben, die nicht in der Lage oder nicht gewillt sind, die öffentliche Web-Oberfläche zu nutzen. Diese portable Softwarevariante ist interessant für zukünftige, kommerzielle Vermarktungsmöglichkeiten in unternehmensinternen Bereichen und für Endanwender. Weiterhin sind die durchaus verschiedenen Einstellungen der Benutzer zu öffentlichen Web-Diensten zu beachten. In dieser Hinsicht ist vor allem der Datenschutz ausschlaggebend für einen Dienst der benutzerdefinierte, private Dokumente verarbeitet. Ein portables Picapica, das auf dem

heimischen Computer arbeitet, ist damit denkbar und vom Benutzer besser kontrollierbar.

Für die Entwicklung des portablen Picapica wurden die im zweiten Kapitel vorgestellten Kriterien angelegt. Das portable Picapica ist durch dieselben funktionalen Grundeigenschaften gekennzeichnet, wie das verteilte System. Diese wurden so angepasst, dass sie effizient auf einem Rechner bewerkstelligt werden können.

Die Portabilität des Softwaresystems ist aufgrund der eingesetzten Programmiersprachen Java im Hinblick auf die Quellcodeportabilität bereits gegeben. Anders sieht es mit den externen Abhängigkeiten zu den zusätzlichen Softwareprodukten aus. Diese Aspekte der Portabilität wurde im Vorfeld charakterisiert und mit den zusätzlichen Softwareanforderungen vereint.

Die Anforderungen an die portable Version des Plagiatanalysesystems stellen sich folgendermaßen dar:

Transportmedien
: Die Software soll mittels CDs oder DVDs an interessierte Anwender verteilt werden. Der Anwender soll frei von Abhängigkeiten sein, die ihm durch das portable System auferlegt werden können. Dabei ist vor allem an softwaretechnische Abhängigkeiten zu denken. Aus diesem Grund, sind alle benötigten externen Programme und zusätzlichen Servertechnologien mitzuliefern.

Plattformen
: Die portable Plagiatanalyse soll unter den bereits genannten Betriebssystemen, Windows, Linux und Mac OS X lauffähig sein. Dabei wird die Priorität vorrangig auf die Windows-Plattform gelegt.

Portabilität und Portierung
: Eine Quellcodeportierung für die geforderten Betriebssysteme ist aufgrund der Programmiersprache nicht nötig. Allerdings muss die Portabilität der externen Software geprüft werden und gegebenenfalls anderweitig gelöst werden.

Das portable Picapica nutzt die Eigenschaften der bestehenden Softwarearchitektur aus, um diesen Anforderungen gerecht zu werden. Damit ist vor allem der strukturelle Aufbau der Software gemeint, denn die weiter oben genannten Komponenten finden auch hier ihre Anwendung. In diesem Zusammenhang bildet das portable Picapica eine weitere Komponente, die die Funktionalität der anderen kombiniert und in einem zusätzlichen, portablen Anwendungskontext einsetzt. Die Architektur des portablen Picapica ist dieselbe, wie die des Web-Dienstes. Dabei bestand die Aufgabe, die unterschiedlichen Server der drei Hauptbereiche Analyse, Kommunikation und Datenaustausch sowie der Benutzerinteraktion in einer Anwendung zu vereinen. Zusätzlich waren die erwähnten externen Applikationen zu berücksichtigen, deren Portabilität Grenzen gesetzt sind. Aus diesen Betrachtungen heraus wurde eine Kompositionsanwendung aus mehreren Komponenten erstellt. In diesem Zusammenhang wurde das portable Plagiatanalysesystem Picapica nach folgenden Methoden implementiert:

- Alle genannten Server sind als ausführbare Einheiten in Threads organisiert. Über eine generische *EmbeddedServer* Schnittstelle, werden die integrierten Server entsprechend angesprochen. Zum Einsatz kommt ein minimaler Servlet-Container,

der sowohl als Server für die Web-Oberfläche, als auch als Datenlager für den Jobserver dient. Der als Message-Queue eingesetzte Datenbankserver ist auch beim portablen Picapica ein MySQL-Server. Alle Server sind überwiegend Java-basiert oder bieten Java-Schnittstellen an Damit entfallen die systemspezifischen, binären Abhängigkeiten der Server. Die Kommunikation kann ungehindert über die bekannten Protokolle stattfinden mit der Ausnahme, dass die Server nur mit anderen Servern, des gleichen Rechners kommunizieren können. Damit ist der Datenbankserver als Message-Queue nur über die lokale Netzwerkschnittstelle erreichbar, genauso wie der Java-Servlet-Container und der Jobserver.

- Eine minimale graphische Oberfläche interagiert mit dem Benutzer im Fehlerfall. Die Oberfläche zeigt dazu die problematischen Ressourcen und macht Vorschläge für die Einstellung. Diese graphische Oberfläche ist nur bei problembehafteten Starts sichtbar und aktiviert ansonsten per Vorgabe die eingebetteten Server. Zusätzlich zeigt ein Icon im Systray der Betriebssystemumgebung an, dass das portable Picapica aktiv ist.
- Da die Funktionalität der Plagiatanalyse unverändert in den portablen Anwendungskontext übernommen wird, sind die zusätzlichen externen Programme zu beachten. Aufgrund der vorrangigen Entwicklungsplattform Windows, sind portable Versionen der externen Software, schnell gefunden. Diese arbeiten ohne vorherige Installation nur im Kontext der portablen Plagiatanalyse. Ebenfalls integriert ist die Java-Laufzeitumgebung und damit die Grundlage für das komplette portable Picapica. Damit existieren Binärpakte für Java, OpenOffice, Ghostgum, Ghostscript und Wget, die unter einer 32Bit-Plattform mit Microsoft Windows ab XP lauffähig sind. Auf gleiche Weise können andere Plattformen bedient werden.
- Das Auslieferungsformat des portablen Picapica ist eine CD oder DVD. Vor der Benutzung ist die Software aus den plattform-spezifischen Archiven, auf einem ausreichend freiem Datenträger zu installieren. Der Speicherplatz ist wegen Beschränkungen in den eingesetzten Bibliotheken und Dritt-Anwendungen notwendig.

Das portable Picapica erfüllt sämtliche, externen Abhängigkeiten indem alle benötigten Softwareelemente mitgeliefert werden. Aus Anwendersicht beschränkt sich die Komplexität der Anwendung, ausschließlich auf die Installation und die Verwendung, der bereits erwähnten Web-basierten Oberfläche von Picapica. Sämtliche Konfigurationsdateien bleiben nach einem ersten Start erhalten und sind bereits mit funktionierenden Standardwerten vorbelegt. Während einer Analyse erfordert es das Verarbeitungsmodell, alle gefundenen Referenzdokumente auf den heimischen Rechner herunterzuladen. Dafür ist temporärer Speicher in größerem Umfang notwendig. Um diesen Ressourcenhunger einzudämmen, werden nach Beendigung einer Analyse, sämtliche Analysedateien, die nicht zur Anzeige in der Web-Oberfläche benötigt werden von der Festplatte gelöscht. Zu dieser automatischen Bereinigung gehören vor allem die während der Analyse angelegten temporären Dateien und Meta-Informationen die von der Datenbank referenziert werden. Zudem behält der Anwender im Sinne des Datenschutzes seine privaten Dokumente auf seinem Computer.

4 Zusammenfassung

Diese Arbeit beschäftigt sich mit der Umsetzung einer portablen Variante des Plagiatanalysesystems Picapica. Um die notwendige Befähigung des Softwaresystems für einen portablen Einsatz zu überblicken, macht einen Rückblick auf die Modellierung und Implementierung von Picapica notwendig. Durch die Erläuterung grundlegender Praktiken während der Softwareentwicklung, werden die Intentionen hinter dem Systemmodell von Picapica deutlich. Damit gibt diese Arbeit gleichzeitig eine Einführung zu Picapica und erläutert Aspekte für eine empfohlene Vorgehensweise beim Softwareentwurf. Diese Aspekte decken das Softwaredesign unter Verwendung etablierter und aktueller Methoden ab, mit Hinblick auf Portabilität und Komponentenorientierung.

Literaturverzeichnis

[Bal98] Balzert, Helmut: *Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung.* Spektrum Akademischer Verlag GmbH, 1998.

[Bal00] Balzert, Helmut: *Lehrbuch der Software-Technik: Software-Entwicklung.* Spektrum Akademischer Verlag GmbH, 2000.

[BSP07] Benno Stein, Sven Meyer zu Eissen und Martin Potthast: *Strategies for Retrieving Plagiarized Documents.* In: Charles Clarke, Norbert Fuhr, Noriko Kando Wessel Kraaij und Arjen de Vries (Herausgeber): *30th Annual International ACM SIGIR Conference*, Seiten 825–826. ACM, July 2007.

[DK75] Deremer, Frank und Hans Kron: *Programming-in-the large versus programming-in-the-small.* In: *Proceedings of the international conference on Reliable software*, Seiten 114–121, New York, NY, USA, 1975. ACM Press.

[Fra94] Franz, M.: *Code-Generation On-The-Fly: A Key to Portable Software.* Doktorarbeit, 1994.

[Gri98] Griffel, Frank: *Componentware.* dpunkt.verlag Heidelberg, 1998.

[GW99] G. Wanner, K.-D. Jäger: *Wiederverwendung von Softwarekomponenten durch entkoppelte Schnittstellen.* OBJEKTspektrum 1/99, 1999.

[Inf08] Inflerwiki: *Softwaretechnik/Script – Inflerwiki*, 2008. [Online; Stand 9. Mai 2008].

[Kop76] Kopetz, Hermann: *Softwarezuverlässigkeit.* Hanser Verlag, 1976.

[Kü94] Küffman, Karin: *Software-Wiederverwendung: Konzeption einer domänenorientierten Architektur.* Viewe & Sohn Verlagsgesellschaft mbH, 1994.

[Moo] Mooney, James D.: *Bringing Portability to the Software Process.*

[MS06] Meyer zu Eissen, Sven und Benno Stein: *Intrinsic Plagiarism Detection.* In: Lalmas, Mounia, Andy MacFarlane, Stefan Rüger, Anastasios Tombros, Theodora Tsikrika und Alexei Yavlinsky (Herausgeber): *Advances in Information Retrieval: Proceedings of the 28th European Conference on IR Research, ECIR 2006, London*, Band 3936 LNCS der Reihe *Lecture Notes in Computer Science*, Seiten 565–569. Springer, 2006.

[Pap08] Papazoglou, Michael P.: *Web Services: Principles and Technology.* Pearson Education Limited, 2008.

[SKS07] Stein, Benno, Moshe Koppel und Efstathios Stamatatos: *Plagiarism Analysis, Authorship Identification, and Near-Duplicate Detection (PAN' 07).* SIGIR Forum, 41(2):68–71, Dezember 2007.

[SM07] Stein, Benno und Sven Meyer zu Eissen: *Fingerprint-based Similarity Search and its Applications.* In: Kremer, Kurt und Wolker Macho (Herausgeber): *Forschung und wissenschaftliches Rechnen 2006*, Seiten 85–98, Göttingen, 2007. Gesellschaft für wissenschaftliche Datenverarbeitung.

[SP06] Stein, Benno und Martin Potthast: *Hashing-basierte Indizierung: Anwendungsszenarien, Theorie und Methoden.* In: *Proceedings of the Workshop Information Retrieval 2006 of the Special Interest Group Information Retrieval (FGIR) in conjunction with Lernen – Wissensentdeckung – Adaptivität 2006 (LWA '06)*, Seiten 159–166, 2006.

[Ste78] Stern, Max: *Some experience in building portable software.* In: *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, Seiten 327–332, Piscataway, NJ, USA, 1978. IEEE Press.

[Ste05] Stein, Benno: *Fuzzy-Fingerprints for Text-Based Information Retrieval.* In: Tochtermann, Klaus und Hermann Maurer (Herausgeber): *Proceedings of the 5th International Conference on Knowledge Management (I-KNOW 05), Graz, Austria*, Journal of Universal Computer Science, Seiten 572–579. Know-Center, Juli 2005.

[TKB78] Tanenbaum, Andrew S., Paul Klint und A. P. Wim Böhm: *Guidelines for Software Portability.* Softw., Pract. Exper., 8(6):681–698, 1978.

[Wik08a] Wikipedia: *Component-based software engineering — Wikipedia, The Free Encyclopedia*, 2008. [Online; Stand 25. April 2008].

[Wik08b] Wikipedia: *Komponente (Software) — Wikipedia, Die freie Enzyklopädie*, 2008. [Online; Stand 25. April 2008].

[Wik08c] Wikipedia: *Modeling language — Wikipedia, The Free Encyclopedia*, 2008. [Online; Stand 22. Mai 2008].

[Wik08d] Wikipedia: *Portabilität (Informatik) — Wikipedia, Die freie Enzyklopädie*, 2008. [Online; Stand 30. April 2008].